凝聚名家技术典范·分享成功IT之路

涵盖 DB2 9.7

循序渐进DB2
——DBA系统管理、运维与应用案例

作序推荐

张挺
前SAP大中国区核心技术中心经理
国内顶尖SAP Basis 顾问

刘晶炜
IBM中国区DB2信息管理技术经理

袁春光
IBM官方资深讲师，咨询顾问
DB2、AIX和WebSphere技术专家



牛新庄 著

清华大学出版社

循序渐进 DB2

——DBA 系统管理、运维与应用案例

牛新庄　著

清华大学出版社
北　京

内 容 简 介

DB2 数据库是 IBM 公司关系型数据库核心产品，在国内以及全球有着广泛的应用。针对 DB2 初学者，本书循序渐进地把 DB2 所涉及的众多概念介绍给大家。客户端连通性、实例、数据库、表空间和缓冲池、数据移动、备份恢复、故障诊断、锁与并发，以及数据库安全都是本书关注的重点。在介绍这些 DB2 对象和概念的同时，作者尽可能从 DBA 日常工作的角度探究 DB2 数据库常规维护工作。本书同时还就表、索引、序列、触发器等数据库对象从应用设计的角度进行了介绍。

本书适合 DB2 的初学者、DB2 开发人员、准备参加 DB2 认证考试的读者以及 DB2 数据库管理人员学习和阅读。

图书在版编目(CIP)数据

循序渐进 DB2——DBA 系统管理、运维与应用案例/牛新庄　著．—北京：清华大学出版社，2009.6

ISBN 978-7-302-20138-0

Ⅰ. 循…　Ⅱ. 牛…　Ⅲ. 关系数据库—数据库管理系统　Ⅳ. TP311.138

中国版本图书馆 CIP 数据核字(2009)第 071577 号

责任编辑：王　军　李维杰
装帧设计：孔祥丰
责任校对：胡雁翎
责任印制：
出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：185×230　　**印　张**：42.75　　**字　数**：880 千字
版　次：2009 年 6 月第 1 版　　**印　次**：2009 年 6 月第 1 次印刷
印　数：1~4000
定　价：80.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：

序　一

DB2 数据库进入中国其实已经很多年，还依稀记得本人安装的第一套基于 DB2 数据库的 SAP 系统还是在 10 多年前的事情了，随着 DB2 在技术上的逐步完善和发展越来越多的企业用户加入了 DB2 的行列。

令人感到遗憾的是，与越来越庞大的 DB2 用户群和逐渐深入的系统应用相比，与之配套的相关中文资料却相当匮乏。除了一些从原版翻译的入门资料外，几乎没有任何全面阐述系统性能调优之类的进阶书籍，面对出现的各种问题，很多用户都只能依靠在网上搜索一些零星的知识点或解决方案，虽然也能暂时地应付一些突发的问题，但很多时候还是处于头痛医头，脚痛医脚的阶段。对 DB2 数据库系统性能优化的整体解决方案缺乏了解。犹如置身于一个巨大迷宫一隅，仅仅靠着身边微弱的烛光艰难前行。

数据库性能调优的需求一直贯穿于整个数据库运行的始终，也是直接关系到以数据库为基础的各种应用软件运行效率。本人从事 SAP 系统性能优化 10 多年，其中很大一部分的工作也是在数据库层面上。深感数据库性能调优的重要性。过去一直苦于没有系统全面的 DB2 方面的资料，只好靠自己慢慢摸索，虽然花费了大量的实践和精力，但效果依然不尽如人意。

一直期盼能有一本理论联系实际，透彻分析数据库工作原理并结合实际案例的工具书来提高工作效率。

纵观当今业内， 我们并不缺少数据库方面的理论宗师，更不缺乏技术娴熟的实践高手。 但同时身兼两大绝学并且将自己的多年积累的实践经验与广大DB2用户分享的，牛新庄博士当属国内第一人。

本书的出版，填补了国内在这方面的空白，书中将各种离散的知识点有机地结合起来并以全新的视角来俯瞰整个数据库的性能问题，使广大 DB2 数据库用户有了一个事半功倍的利器。

本书完全应该成为DB2数据库从业者以及相关技术人员人手一本的工具书。

前 SAP 大中国区核心技术中心经理

国内顶尖SAP Basis顾问

上海建功思域信息科技有限公司　董事总经理

在 SAP 技术领域有着极为丰富的实践经验和专业知识背景，尤其擅长大型系统架构设计和系统性能调优。

张　挺

2009 年 5 月

序　二

这些年我负责中国 DB2 的售前技术团队，认识牛新庄已经好多年了。其中印象很深的是 2006 年与他的几次交流，我们讨论了许多数据库的实用技术，用户在数据库管理和应用方面的主要挑战。他丰富的实战经验给我留下了极深的印象。那时他正在参加首届中国数据库工程师大赛，最终获得了最高的杰出数据库工程师大奖。

新庄是国内较早使用 DB2 的技术人员。他从 1999 年开始使用 DB2 V5.2，后来在工作中又学习了 AIX、WebSphere、CICS 和 MQ 等 IBM 技术，对 Informix 和 Oracle 等数据库也有非常深的理解。这些年他作为 IBM 培训部的资深认证讲师在国内讲解数据库技术，推动 DB2 在国内的传播。从 2001 年至今，新庄作为独立咨询顾问往返于国内大中城市，在金融六大行(工农中建交招)，农信，证券(国泰君安、海通、大通等)，电力(江苏电力公司、云南电力公司、山东电力公司等)，保险(中国人寿、信诚、平安等)，电信，邮政，移动(北京移动、上海移动、江苏移动、广东移动、天津移动、湖南移动、西藏移动、新疆移动、山东移动、吉林移动等)，青岛海尔，云南红塔，中远集团，宝钢等行业和国内中小企业之间做数据库架构设计、维护、问题诊断和性能调优。

其深厚的产品知识和丰富的阅历和经验使得他在对一些疑难问题的判断和处理上有独到的见解。他往往能够跳出固有的框架从一个广阔的视角来认识和分析，并通过多年积累的方法论逐步排查，最终找到解决的方法。这一点在许多大型用户的案例中都得到了有效的应证，这些年我听到了很多大型用户对新庄技术的高度认可。

DB2 的学习资料在其信息文档、网络中有很多，然而由拥有丰富实践应用经验的专家来总结的 DB2 书籍相对较少。新庄写的这套书特点是注重实用，内容由浅及深，涵盖 DB2 的管理，运行维护，应用开发，内核及架构的剖析，以及性能调整和优化，我认为本套书将一系列相关的分散知识点真正形成了一个知识面。用好 DB2 数据库实际上涉及很多方面，不仅仅是数据库本身，而且需要考虑操作系统，存储规划，数据模型设计，应用开发设计，数据库的合理配置和运行监控等一系列相关的内容。

本套书不仅从产品的角度来介绍 DB2，而且从实战的角度来剖析基于 DB2 设计和应用。一个好的应用系统应该考虑哪些问题，如何有效管理维护好 DB2 系统，常见的故障如何排查及解决，在应用开发中如何有效使用 DB2 的特性。本书系统性地总结了 DB2 的发

展历史，从一个系统构建生命周期的角度介绍了 DB2 数据库的安装、使用、开发、管理、运行、调优的全过程。深入 DB2 内部剖析其核心架构，结合案例分享实战应用调优的经验。尤其是性能调优和优化这本书浓缩了新庄自己在应用 DB2 的心路历程，内容覆盖了系统的整体设计规划，DB2 与性能相关的内部核心技术和架构，关键的相关应用设计要点，以及稳定运行监控所应考虑的内容。最难得的是作者分享了多年积累的 DB2 性能调优案例，使读者有可能在实际的环境中去了解解决复杂问题的思路，将基本的理论和技术与实战进行结合。

这套书不仅是新庄 10 年 DB2 应用经验的总结，更是他 10 年对 DB2 数据库的付出和钻研的结晶。今天国内已有越来越多的技术人员在使用 DB2，我相信这套书能对学习和使用 DB2 提供很大的帮助。希望它能成为您 DB2 旅程上的一个朋友，为您答疑解惑，点亮您前进的道路。

IBM 中国区 DB2 信息管理技术经理
刘晶炜
2009 年 1 月　于北京

序　三

与牛新庄相识相知也有十余年了，牛新庄作为数据库技术的资深技术专家有着非常深厚的理论功底和极其丰富的实践经验。他是国内较早使用 DB2 的技术人员，同时对 Informix 和 Oracle 等数据库也有非常好的掌握，同时还涉猎了很多其他的技术，如 WebSphere、MQ、CICS 等等。这些年他作为 IBM 培训部的资深讲师在国内讲解数据库技术并从事 DB2 的各种咨询顾问工作。从 2001 年至今，新庄作为独立咨询顾问为国内大量的大中型企业提供了高端的数据库技术咨询工作，得到了这些客户一致的认可和极高的评价。这么多年来，尤其让我佩服的是他长期以来、多年如一日的持续学习的平和心态和对技术孜孜不倦的探索精神所构成的勤奋。我想这种勤奋才是其成功的关键，而且也应该是最值得我等技术人员学习和汲取的精华所在。

我开始使用 DB2 数据库已经是将近十年前的事情，作为 IBM 的讲师也已经有将近 5 年的时间了，期间的感受颇多，在不断的学习、实践和教学的过程中，非常大的一个感受就是国内 DB2 技术书籍非常的匮乏。多年来这一情况基本上没有太大改观。由于中文学习资料的缺乏导致很多刚刚接触 DB2 的初学者经常会望而却步，我接触到的很多客户和学员都不约而同地由此感叹。这样就使广大的数据库技术人员觉得 DB2 距离"民间"总是有段较大的距离，认为它比较神秘而不普及。而事实上从 DB2 所应用的领域和行业来看又并非如此，我们现在在各个领域和各行各业基本上都能看到 DB2 的应用。

由牛新庄编写的这套 DB2 数据库管理和开发丛书正是满足了国内广大 DB2 数据库爱好者的这一需求。而且这套丛书针对的读者群非常的广泛：有专门针对 DB2 初学者的《循序渐进 DB2——DBA 系统管理、运维与应用案例》，有专门适合于 DB2 资深管理人员的《深入解析 DB2》，还有专门用于帮助有经验的 DB2 管理员如何调优数据库的《DB2 数据库性能调整和优化》，另外还有从开发人员角度来解决 DB2 问题的《DB2 应用开发实战指导》。这套书的知识体系几乎全面覆盖了 DB2 数据库常用技术的所有领域，在倾注了牛新庄多年的技术积累和丰富的实战经验之后更显得此套丛书的弥足珍贵。

《循序渐进 DB2——DBA 系统管理、运维与应用案例》一书包含了一个 DB2 的 DBA 对数据库日常管理的主要内容。这本书非常适合于 DB2 技术的初学者和具有一定经验的 DB2 管理员。这本书从 DB2 数据库的安装到各种数据库对象的创建和管理，从 DB2 数据库的逻辑设计到物理设计需要注意的各种问题，从 DB2 数据库中数据的移动到数据库的备份和恢复，从 DB2 数据库的监控到 DB2 的故障诊断和处理，从 DB2 数据库性能的监

控调整到日常需要做的一些琐碎维护工作，最后还介绍了一些 DBA 能够派得上用场的实用工具。这些内容无一不是切合工作的实际情况，从 DBA 的角度全面考量、精心编排的结晶。

作为战斗在一线的工程师和普及数据库知识的讲师，我深感《循序渐进 DB2——DBA 系统管理、运维与应用案例》一书具有非常强的实用性。通过名称我们就能发现，这本书除了知识内容以外还包含了大量的实际案例，这些案例都是牛新庄从一些经验和教训中总结出来的成果，这些内容实在是弥足珍贵，更是让 DBA 有机会占在巨人的肩膀上处理实际生产中的工作和问题。

这本书适合于绝大多数的 DB2 初学者和一般的 DB2 管理人员，而且我相信这本书除了会带给读者 DB2 的知识之外还能让读者获得难得的经验。除了这本书本身带给我们的知识体验之外，如果我们进一步地能够被作者牛新庄的学习态度和踏实精神所感染，经历一次精神的体验，而走出一条自己的技术之路，那么这本书就真正体现了其价值。

IBM 官方资深讲师，咨询顾问，DB2、AIX 和 WebSphere 技术专家
袁春光
2009 年 5 月

前　　言

DB2 数据库是 IBM 公司关系型数据库核心产品，在国内以及全球有着广泛的应用。针对 DB2 初学者，本书循序渐进地把 DB2 所涉及的众多概念介绍给大家。客户端连通性、实例、数据库、表空间和缓冲池、数据移动、备份恢复、故障诊断、锁与并发以及数据库安全都是本书关注的重点。在介绍这些 DB2 对象和概念的同时，作者尽可能从 DBA 日常工作的角度探究 DB2 数据库常规维护工作。本书同时还就表、索引、序列、触发器等数据库对象从应用设计的角度进行了介绍。本书适合 DB2 的初学者、DB2 开发人员、准备参加 DB2 认证考试的读者以及 DB2 数据库管理人员学习和阅读。

本书结构

本书共 15 章，具体结构如下。

第 1 章：DB2 安装配置。在这一章中，除介绍初学者比较熟悉的 Windows 安装外，还花费了比较多的篇幅介绍了在 Unix/Linux 环境下的安装。这主要是因为作者碰到的 DB2 生产环境几乎都是在 Unix/Linux 环境下，而在 Unix/Linux 环境下的安装 DB2 时涉及到的准备工作又远大于 Windows 环境下。

第 2 章：创建实例和管理服务器。与其他数据库系统类似，DB2 中也存在实例概念，主要对应着 DB2 二进制代码。而管理服务器则是 DB2 中特有的，用于帮助 DBA 对远程主机上的多个实例进行控制。本章中详细介绍了实例的创建、删除、配置以及相关的操作系统环境变量等，对管理服务器由于生产实践中使用较少则进行了简单介绍。

第 3 章：创建数据库和表空间。本章中介绍了 DB2 数据库的存储模型，创建数据库命令的具体选项对后继工作的影响。本章重点介绍了 DB2 数据库表空间的管理类型，并指出不同类型之间的优缺点。在表空间部分，本章还讲述了影响表空间性能的所有选项，如预取大小、扩展大小等，同时又指出操作系统 IO 设置对表空间性能影响。与表空间关联的是缓冲池，本章给出了缓冲池的设计、维护原则。

第 4 章：访问数据库。本章介绍了如何配置 DB2 服务器与客户端，使得客户端能够访问服务器上的数据。本章介绍了 DB2 命令行工具 CLP 使用，同时也讲述了在客户端上如何通过各种图形工具配置到服务器的连通性。在这些基础上，本章给出了 DB2 节点目录、数据库目录、本地数据库目录之间的相互关系与区别。

第 5 章：创建数据库对象。本章中介绍了常见 DB2 对象的维护方法，重点讲述了数据库中最重要的对象——表的设计考虑。同时本章也介绍了如何使用索引、序列提高性能。

第 6 章：数据移动。在创建完表等对象后，DBA 下一步工作就是向表中填充数据。几乎所有系统的构建都涉及数据移动。本章介绍了从数据库中导出数据、向数据库导入数据，重点讲述了 DB2 效率非常高的数据移动工具 LOAD。对 LOAD 工具，讲述了如何在线 LOAD、如何监视 LOAD、LOAD 性能提高选项、LOAD 异常处理等。在本章中，作者总结了数据移动中经常出现的问题，并给出了相关解决办法。最后，本章介绍了集成数据移动工具 db2move 和数据字典抽取工具 db2look 的使用。

第 7 章：数据库备份与恢复。本章中介绍了数据库系统通常碰到的几种备份恢复类型，并指出 DB2 如何配置日志以支持这些类型的。本章中描述了各种情况下如何重建数据库，同时给出了监控 DB2 数据库备份、恢复进度的方法，以及如何优化备份恢复的速度。

第 8 章：DB2 故障诊断。数据库系统难免会出现各种各样的故障，DB2 中有着一套完整的故障诊断机制。本章重点介绍了故障诊断中最重要的日志文件 db2diag.log 文件的格式，并且对各种格式条目给出详细解释。故障诊断中工具的使用是必不可少的，本章介绍了几个常用工具的使用方法，如 db2pd、db2level、db2ls、db2support 等。

第 9 章：DB2 性能监控。DB2 数据库给出了多种手段用于监控数据库内部运行情况，如事件监控、快照监控、动态性能视图等。本章主要介绍了实践中使用较多的快照监控，给出了许多生产中的实际案例。

第 10 章：锁和并发。数据库系统设计用于并发支持大量用户连接到系统操作数据，锁在这里起到了关键性作用，特别是在 OLTP 系统中对性能有着至关重要的影响。本章首先介绍了通用的事务概念，并指出 DB2 在用户读取、写入数据时的加锁策略、锁模式、兼容性等。本章还介绍了在并发控制中常碰到的 4 种数据异常现象，同时讲述了 DB2 如何使用锁克服这些问题的。

第 11 章：数据库运行维护。一个数据库系统建设完成后，DBA 是否就可以高枕无忧了呢？随着数据量的增加、用户数的增多，性能可能越来越差。这时需要 DBA 进行运行维护工作，本章重点关注这些内容，包括统计信息更新、表和索引碎片整理、包重新绑定等。持续进行这些维护工作有助于避免数据库系统性能下降。

第 12 章：数据库常用工具。本章介绍了 DBA 在日常工作中经常使用的各种工具，如性能解释工具、数据设计建议工具、基准测试工具、数据库一致性检查工具等。熟练掌握这些工具，对 DBA 而言犹如利器在手。

第 13 章：数据库安全。DB2 数据库安全控制包括身份认证、权限、特权三个层次。身份认证控制着谁访问数据库，权限和特权则控制着能访问什么数据，两者只是控制粒度上存在差别。本章介绍了 DB2 何时进行身份认证以及进行什么认证，在实例、数据库上两

个如何授予、撤销权限，以及特权的授予与撤销。

第 14 章：DBA 日常维护。DBA 的职责是保证数据库稳定、高效运行，除了正常的运行维护外，DBA 还经常碰到各种其他问题，本章主要介绍了作者在日常工作进行的维护工作。本章首先介绍了如何对 DB2 数据库健康性检查以及检查涉及的各个方面，然后给出了找出各种类型的 TOP10 的 SQL 语句方法。

第 15 章：DB2 常见问题总结。本章是对全书的总结，是作者多年来在各种环境下碰到实际问题的共享。

致谢

本书在出版的过程中得到了清华大学出版社王军编辑的大力支持！这套 DB2 书籍从选题、审稿到出版无不得到他的热心帮助，在此致以深深的谢意！

感谢我的好兄弟骆洪青和袁春光，他们审核了书中的大部分章节。同时也感谢中信银行的胡瑞娟、苏兰芳和我的师弟林春，他们审核了部分章节并从用户的角度给我提出了很多宝贵的建议！

最后，谨以此书献给我慈爱的母亲，母亲从小就教育我努力、正直、踏实和勤奋。正是由于母亲的影响和教育才有了我今天的一点微小的成绩。

新庄

于 2009 年 4 月

目　　录



DB2 安装配置

DB2 是 IBM 于 1983 年推出的第一款面向大型企业的商业化关系数据库管理系统。80 年代初 DB2 的发展重点放在大型的主机平台，从 80 年代中期到 90 年代初 DB2 已发展到中型机、小型机以及微机平台。DB2 的诞生不仅促进了与关系数据库概念相关的数学和科学的发展，还创造性地开发出一个极具影响力的全新软件类型。今天，DB2 已经发展成为 IBM 信息管理软件组合的重要组成部分。IBM 信息管理软件组合是 IBM 5 大中间件品牌之一。在 IBM 信息随需应变策略和体系结构中，DB2 扮演数据服务器的角色，并且已经发展成同时支持传统关系数据和 XML 的混合型数据服务器。传承 IBM 数据库的优良传统并具有突破性的数据库产品 DB2 V9 于 2006 年底问世，它以先进的技术和理念将数据库技术引入一个以 XML 应用为主导的新纪元。

DB2 数据库产品及解决方案广泛应用在金融、电信、制造、零售、保险等行业及政府机关，以数据库技术创新帮助客户实现更大价值，以技术创新推动商业模式的变革和不断发展。

本章主要讲解如下内容：

- DB2 数据库概述
- DB2 数据库安装和配置
- DB2 数据库体系结构

1.1 DB2 数据库概述

1.1.1 DB2 发展历史

我们都知道DB2是数据库的一种，那么在我们开始学习DB2之前，先让我们了解一下数据库的发展历史。

1. 数据库发展历史

在没有数据库之前，人们靠什么来记录数据呢？最早是靠文件，但是用文件记录有很多缺点，例如不易保存和共享等。而数据库的出现可以解决这些问题。数据库的历史可以追溯到 40 多年前，当时计算机开始广泛地应用于数据管理，对数据的共享提出了越来越高的要求。传统的文件方式已经不能满足人们的需要。能够统一管理和共享数据的数据库管理系统(DBMS)应运而生。数据模型是数据库系统的核心和基础，各种 DBMS 软件都是基于某种数据模型的。所以通常也按照数据模型的特点将传统数据库系统分成网状数据库、层次数据库和关系数据库 3 类。最早出现的是网状 DBMS，1961 年通用电气公司(General Electric)的 Charles Bachman 成功地开发出世界上第一个网状 DBMS，也是第一个数据库管理系统——集成数据存储(Integrated DataStore，IDS)，奠定了网状数据库的基础，并在当时得到了广泛的发行和应用。IDS 具有数据模式和日志的特征。但它只能在 GE 主机上运行，并且数据库只有一个文件，数据库所有的表必须通过手工编码来生成。网状数据库模型对于层次和非层次结构的事物都能比较自然地模拟，在关系数据库出现之前网状 DBMS 要比层次 DBMS 用得普遍。在数据库发展史上，网状数据库占有重要地位。层次型 DBMS 是紧随网络型数据库而出现的。最著名最典型的层次数据库系统是 IBM 公司在 1968 年开发的 IMS(Information Management System)——一种适合其主机的层次数据库。这是 IBM 公司研制的最早的大型数据库系统程序产品。从 60 年代末产生起，如今已经发展到 IMS V10，提供对群集、N 路数据共享、消息队列共享等先进特性的支持。这个具有 30 年历史的数据库产品在如今的 WWW 应用连接、商务智能应用中仍扮演着新的角色。目前国内的 4 大银行在主机上仍然在使用 IMS 数据库。

关系数据库

关系数据库的由来：网状数据库和层次数据库已经很好地解决了数据的集中和共享问题，但是在数据独立性和抽象级别上仍有很大欠缺。用户在对这两种数据库进行存取时，仍然需要明确数据的存储结构，指出存取路径。而后来出现的关系数据库较好地解决了这些问题。1970 年，IBM 的研究员 E.F.Codd 博士在刊物《Communication of the ACM》上发表了一篇名为“A Relational Model of Data for Large Shared Data Banks”的论文，提出了关系模型的概念，奠定了关系模型的理论基础。尽管之前在 1968 年 Childs 已经提出了面向集合的模型，然而这篇论文被普遍认为是数据库系统历史上具有划时代意义的里程碑。Codd 的心愿是为数据库建立一个优美的数据模型。后来 Codd 又陆续发表多篇文章，论述了范式理论和衡量关系系统的 12 条标准，用数学理论奠定了关系数据库的基础。IBM 的 Ray Boyce 和 Don Chamberlin 将 Codd 关系数据库的 12 条准则的数学定义以简单的关键字语法表现出来，里程碑式地提出了 SQL(Structured Query Language)语言。关系模型有严格

的数学基础，抽象级别比较高，而且简单清晰，便于理解和使用。但是当时也有人认为关系模型是理想化的数据模型，用来实现 DBMS 是不现实的，尤其担心关系数据库的性能难以接受，更有人视其为当时正在进行中的网状数据库规范化工作的严重威胁。为了促进对问题的理解，1974 年 ACM 牵头组织了一次研讨会，会上开展了一场分别以 Codd 和 Bachman 为首的支持和反对关系数据库两派之间的辩论。这次著名的辩论推动了关系数据库的发展，使其最终成为现代数据库产品的主流。而 Oracle 公司在 1979 年开发了第一个商用 SQL 关系数据库管理系统。关系型数据库系统以关系代数为坚实的理论基础，经过几十年的发展和实际应用，技术越来越成熟和完善。其代表产品有 Oracle、DB2、SQL Server 以及 Informix、Sybase 等等。

面向对象数据库

面向对象数据库：随着信息技术和市场的发展，人们发现关系型数据库系统虽然技术很成熟，但其局限性也是显而易见的——它能很好地处理所谓的“表格型数据”，却对技术界出现的越来越多的复杂类型的数据无能为力。90年代以后，技术界一直在研究和寻求新型数据库系统。但在什么是新型数据库系统的发展方向的问题上，产业界一度是相当困惑的。受当时技术风潮的影响，在相当一段时间内，人们把大量的精力花在研究“面向对象的数据库系统(Object-Oriented Database)”或简称“OO数据库系统”。值得一提的是，美国 Stonebraker教授提出的面向对象的关系型数据库理论曾一度受到产业界的青睐。而 Stonebraker本人也在当时被Informix花大价钱聘为技术总负责人。然而，数年的发展表明，面向对象的关系型数据库系统产品的市场发展的情况并不理想。理论上的完美性并没有带来市场的热烈反应。其不成功的主要原因在于，这种数据库产品的主要设计思想是企图用新型数据库系统来取代现有的数据库系统。这对许多已经运用数据库系统多年并积累了大量工作数据的客户，尤其是大客户来说，是无法承受新旧数据间的转换而带来的巨大工作量及巨额开支的。另外，面向对象的关系型数据库系统使查询语言变得极其复杂，从而使得无论是数据库的开发商还是应用客户都视其复杂的应用技术为畏途。

混合数据库

IBM 经过多年的积累和持续创新，在 2006 年年底率先推出了第一个直接支持 XML 的混合数据服务器——IBM DB2 V9(代号为 Viper)。

IBM DB2 V9 提供了与以前版本非常不同的体系结构，它通过提供新的查询语言、新的存储技术、新的索引技术和支持 XML 数据及其固有层次结构的特性，使得 IBM DB2 V9 成为 IBM 的第一个“混合型”(即多结构)数据库管理系统。除了支持表数据模型外，DB2 还支持 XML 文档和消息中固有的层次化数据模型。用户可以在一个表中自由地混合存储传统 SQL 数据和最新的 XML 数据。还可以使用 SQL(如果愿意，可以加上 XML 扩展)和

XQuery(新出现的 XML 数据查询标准)来查询和处理这两种形式的数据。在经过实践检验的数据库管理基础设施上进行扩展，IBM 为 DB2 V9 用户提供了同时处理关系数据和 XML 数据的强大支持。新的基于 XML 的应用程序使用 DB2 可以无缝地访问关系数据库资源，拥有企业级 XML 应用的访问能力。DB2 在所有特性/接口中都整合了对 XML 数据的支持，而 DB2 对 XML 的“固有”支持是在对其他技术的现有支持之外提供的，SQL、表格数据结构和各种 DBMS 现有特性仍然能够获得最好的支持。因此，用户可以用一个数据库对象同时管理“传统的”SQL 数据和 XML 文档。

为了高效地管理传统 SQL 数据类型和 XML 数据，DB2 包含两种不同的存储机制。但是，一定要注意，给定数据类型所用的底层存储机制对于应用程序是透明的。换句话说，应用程序不需要显性地指定要使用的存储机制，也不需要管理存储的物理细节，比如如何将 XML 文档的各个部分拆分到多个数据库页上。系统会自动采用适合目标数据的格式，应用程序自然而然地获得存储和查询数据方面的运行时性能优势。DB2 V9 的存储模型如图 1-1。

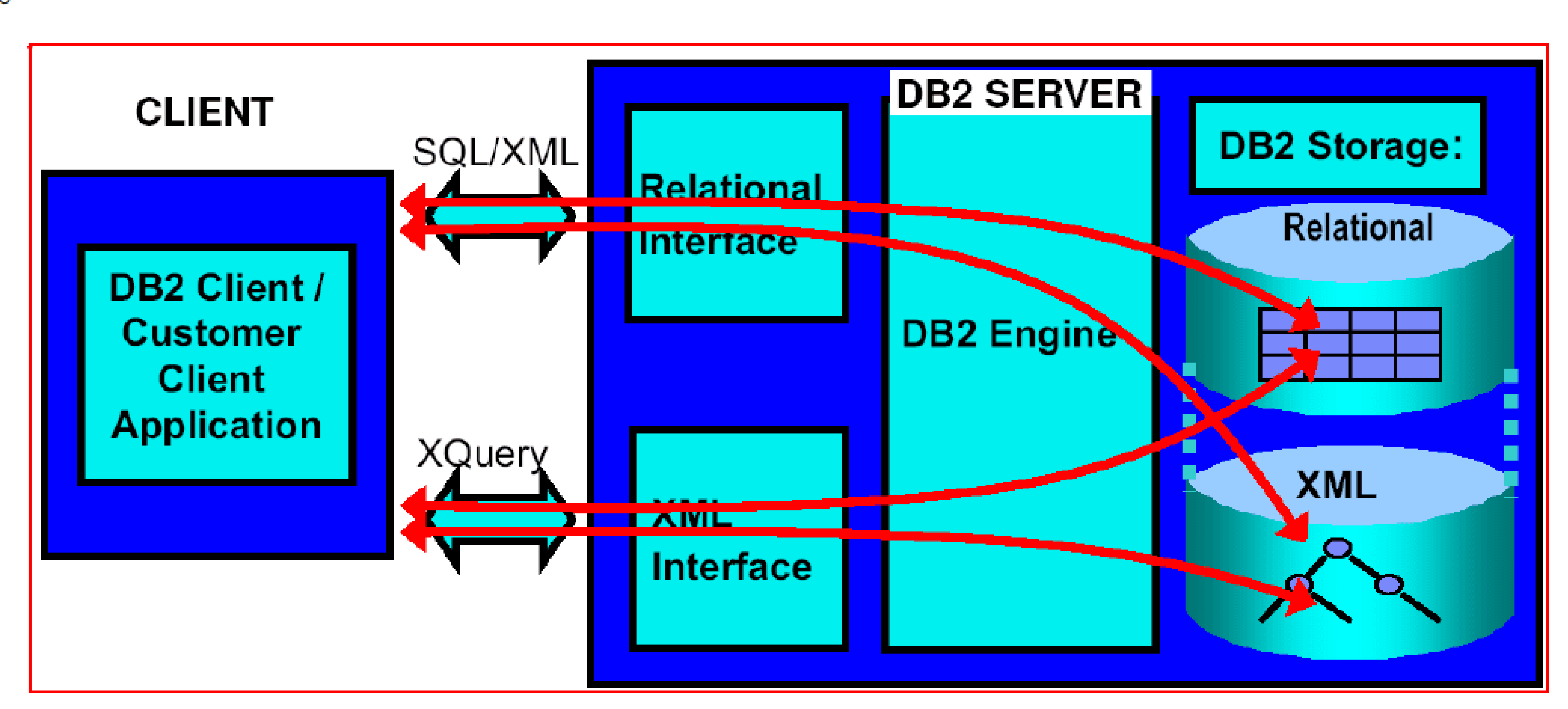


图 1-1　DB2 V9 的存储模型

2. 数据库大事年鉴

- 1969：IBM 开发的层次数据库 IMS 诞生。这是 IBM 公司的第一代数据库，也叫 DB1。
- 1970：IBM 的 E.F.Codd 发表了关于关系数据库技术的第一篇论文“基于大型共享数据库的数据关系模型”。
- 1973：在 IBM 研究中心确立了 System R 项目，建立关系数据库管理系统。

- 1974：IBM 的 Don Chamberlin 和 Ray Boyce 出版了“EQUEL：结构化英语查询语言”，成为 SQL 标准定义的基础。
- 1975：IBM 的 Don Chamberlin 和 Morton Astrahan 的论文“结构化英语查询语言的实现”阐述作为 System R 一部分的第一个 SQL 实现。
- 1976：IBM System R 团队出版了“System R：数据库管理的关系方法”，这篇论文阐述了他们的关系数据库原型。IBM 的 Jim Gray 发表“共享数据库中的锁粒度和结合度”，阐述数据库事务和结合度的正式定义，这是数据库并发理论的基础。
- 1979：IBM 的 Pat Selinger 在她的论文“关系数据库管理系统中的访问路径选择”中阐述行业的第一个关系查询优化器，这是 DB2 数据库优化器的雏形。
- 1979：Oracle 公司开发了第一个商用 SQL 关系数据库管理系统。
- 1980：在一家早期的 S-100/CP/M 公司 Cromemco 工作的 Roger Sippl 和 Laura King 开发了一个基于 ISAM 技术的小型的关系数据库，作为一个报表记录器软件的一部分。1980 年，Sippl 和 King 离开 Cromemco 去开发关系数据库系统(RDS)。1981 年发布了他们自己的一个产品：Informix(INFORMation on unIX)。
- 1982：IBM 为 VSE/VM 发布 SQL/DS，作为第一个商业用途的关系数据库(带有基于 System R 的 SQL 接口)。SQL/DS 是 DB2 数据库的前身。
- 1984：Sybase 公司成立，公司名称“Sybase”取自“system”和“database”相结合的含义。Sybase 公司的创始人之一 Bob Epstein 是 Ingres 大学版(与 System/R 同时期的关系数据库模型产品)的主要设计人员。Sybase 公司的第一个关系数据库产品是 1987 年 5 月推出的 Sybase SQLServer1.0。Sybase 首先提出 Client/Server 数据库体系结构的思想，并率先在 Sybase SQLServer 中实现。
- 1988：SQL Server 由微软与 Sybase 共同开发，最早运行于 OS/2 平台。所以您会发现 SQL Server 和 Sybase 数据库早期的架构基本是一样的。直到 SQL Server 2005 以后，微软才做了大幅度改动。

3. DB2 发展历史

- 1983 年，发布 Database2 (DB2) for MVS，内部代号为“Eagle”，于是 DB2 正式诞生，相对于 IBM 的第一代层次数据库(DB1)而言这是第二代数据库，所以叫 DB2。
- 1986 年，System/38 V7 发布，首次配置查询优化器，能对应用的存取计划进行优化。这是今天 DB2 强大优化器的雏形。这时的 DB2 仍然只能运行在大型机上。

- 1987 年，IBM 推出 OS/2 V1.0 扩展版本，该版本带有关系数据库能力，这是 OS/2、Unix 和 Windows 上 DB2 的先驱。DB2 完成了从大型机到 OS/2 的扩展，进入微机领域。
- 1988 年，在集成 RDBMS 的新 AS/400 服务器上宣布并推出 SQL/400。发布 SQL/400，为 AS/400 服务器提供 SQL 支持，这是 DB2 for i5/OS 的前身。
- 1989 年，IBM 定义 Common SQL 和 IBM 分布式关系数据库体系结构(DRDA)在所有 IBM RDBMS 产品中实现。DRDA 实现开放平台到大型机和 AS/400 的访问。
- 1992 年，DB2 for OS/2 V1 和 DB2 for RS/6000 V1 推出，这是第一次在 Intel 和 UNIX 平台上推出 DB2 产品。
- 1993 年，IBM 宣布并推出 DB2 for OS/2 V1(DB2/2)和 DB2 for AIX V1(DB2/6000)。
- 1994 年，IBM 宣布针对 RS/6000 SP2 平台的 DB2 并行版本 V1，这标志着 IBM 的群集、高扩展性体系结构(专注于复杂查询工作的大型数据仓库)的诞生。IBM 开始将 DB2 V1 扩展到非 IBM UNIX 平台，发布 HP-UX 和 Sun-Solaris 平台上的 DB2 CommonServer 技术。
- 1994 年，DB2 实现了 HP UNIX 和 Solaris 版本。也是在 1994 年，AIX 平台上 DB2 开始支持对象型数据。
- 1995 年，DB2 开始支持 Windows、UNIX 等多个平台(这是标志性的一年)。
- 1996 年，DB2 正式更名为 DB2 UDB 通用数据库。
- 1997 年，DB2 UDB for UNIX/Windows/OS2 同时发布。
- 1998 年，DB2 开始支持 SCO UNIXware。
- 1998 年，DB2 UDB5.2 发布。
- 1999 年，DB2 发布 Linux 平台版本。
- 2000 年，DB2 支持 XML 扩展。
- 2001 年，IBM 完成对 Informix 数据库的收购。
- 2002 年，基于自我管理和资源调节(SMART)技术(IBM 的自主计算发端的一个元素)的 DB2 V8.1 出现。
- 2006 年，DB2 V9 出现，这是一个划时代的革命性的产品，是第一个混合模式(关系型、层次型)数据库，既有关系模型，又有直接支持 XML 的层次模型。IBM 为 Linux、UNIX 和 Windows 引入 DB2 V9(代码名称为“Viper”)，这是第一台启用常规关系数据和 XML 数据管理的混合的数据服务器(无需将 XML 重新格式化或放入数据库内的大对象中)。

1.1.2　DB2 版本和平台支持

1. DB2 平台支持

如图 1-2 所示，DB2 产品几乎覆盖了当前所有流行的硬件和操作系统平台。在大型机操作系统上，有 DB2 for z/OS (DB2 for OS/390、DB2 for MVS/ESA、DB2 for VM/VSE)利用了 System z 服务器上的硬件耦合器(Coupling Facility)，因此与使用“shared-nothing”方式的 DB2 luw 相反，它采用“shared-everything”的方式。

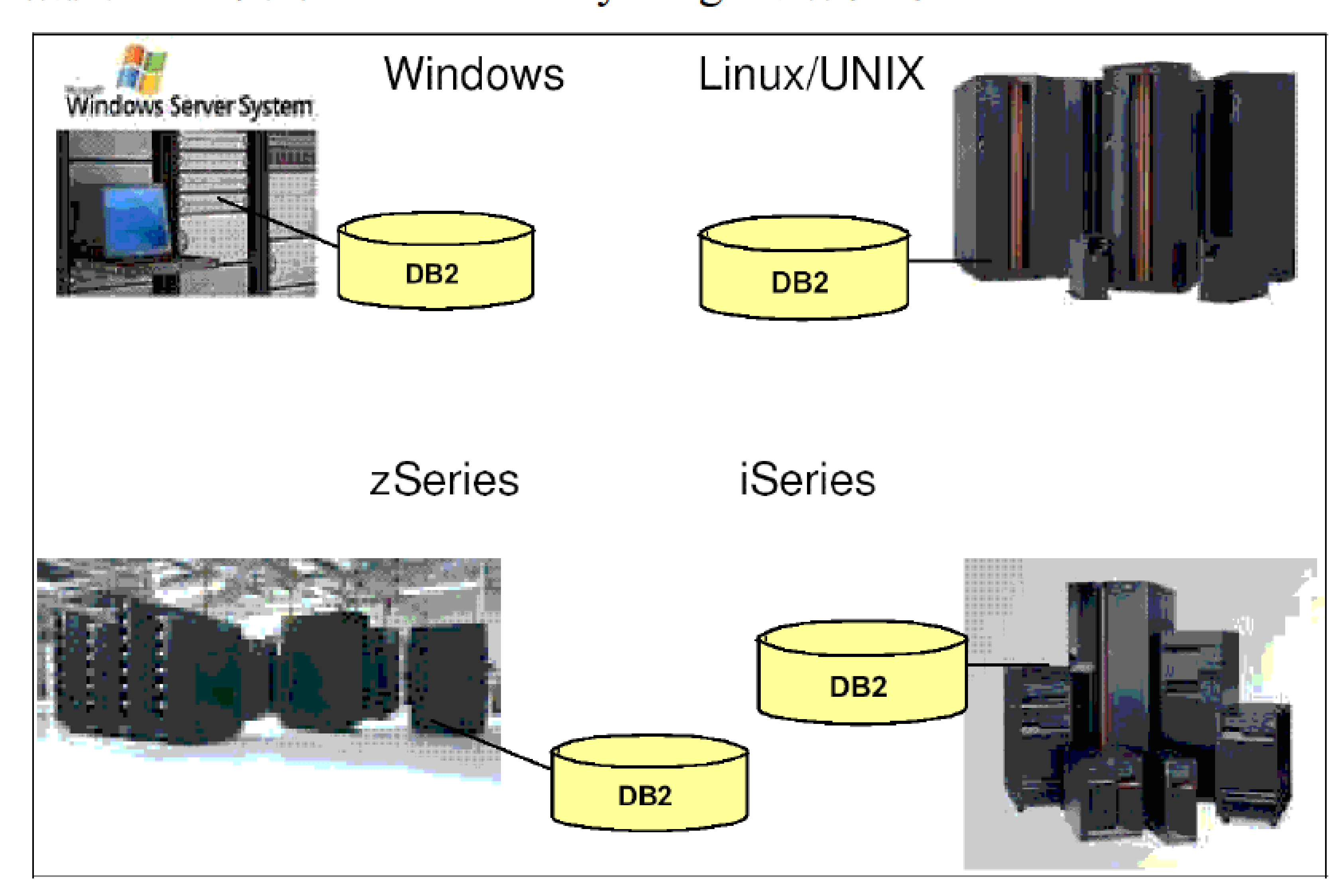


图 1-2　DB2 支持的平台

在由 IBM 公司设计的中型机上(AS/400)，有 DB2 for System i，DB2 已经嵌入在 i5/OS 操作系统中，成为其不可分割的一部分。DB2 对 UNIX 操作系统的支持同样十分广泛，可以在 AIX、HP-UX、Solaris 等多种系统上找到其相应的版本。

在 PC 操作系统上，DB2 可以对 Window 9x、Windows NT、Windows XP 以及 Windows Vista 等多种操作系统提供支持。同样，DB2 从版本 6.1 后也增加了对 Linux 的支持。

以上我们所提到的只是 DB2 服务器所能运行的平台，DB2 的客户端所能支持的平台更为广泛，除了以上提到的所有平台之外，DB2 的客户端还能运行在 DOS、Windows 3.x、Mac OS 以及 SGI 公司的 IRIS 系统之上。综上所述，DB2 数据库服务器对平台的支持主要有 3 种：DB2 for z/OS 大型机平台、DB2 for i5/OS 中型机平台和 DB2 for luw(DB2 for Linux，DB2 for UNIX 和 DB2 for Windows)开放平台。最早的 DB2 产品是 DB2 for MVS/ESA，以后的产品设计都延续了它的基本结构及关键算法，保证了不同系统之间的可移植性、可扩展性和互操作性。但是，由于不同操作系统之间存在着不小的差异，DB2 系列产品还针对相应的平台进行了一定的优化，以适应各个操作系统的特性。在本书我们主要讲解 DB2 for

luw 平台。

注意：
以后在本书中我们要提及的 DB2 如无特别说明指的是 DB2 for luw 平台。

2. 版本支持

DB2 产品除了能够对各种硬件和操作系统平台进行支持之外，DB2 V9 提供了适于所有企业的数据管理解决方案。为了适应不同规模企业和用户的需要，DB2 提供了不同级别的产品，对小到个人用户、大到跨国企业的不同需求提供支持。没有其他数据库管理系统能够在性能、可用性、可伸缩性和可管理性方面达到 DB2V9 的水平。

DB2 有不同的版本，每种版本适合不同需求的用户。图 1-3 显示了所有可用的 DB2 分发版本。从图中可以看出 DB2 的每个高版本都包含低一级版本的所有功能和特性，并添加了新的特性和功能。Linux、UNIX 和 Windows(luw 平台)上的代码有大约 90%是相同的，在每种操作系统上有 10%的专用代码，用于使数据库与底层操作系统紧密地集成。例如，使用 AIX 上的 JFS2 文件系统或 Windows 上的 NTFS 文件系统。

以下是对 DB2 V9 版本中不同级别产品的特点介绍。

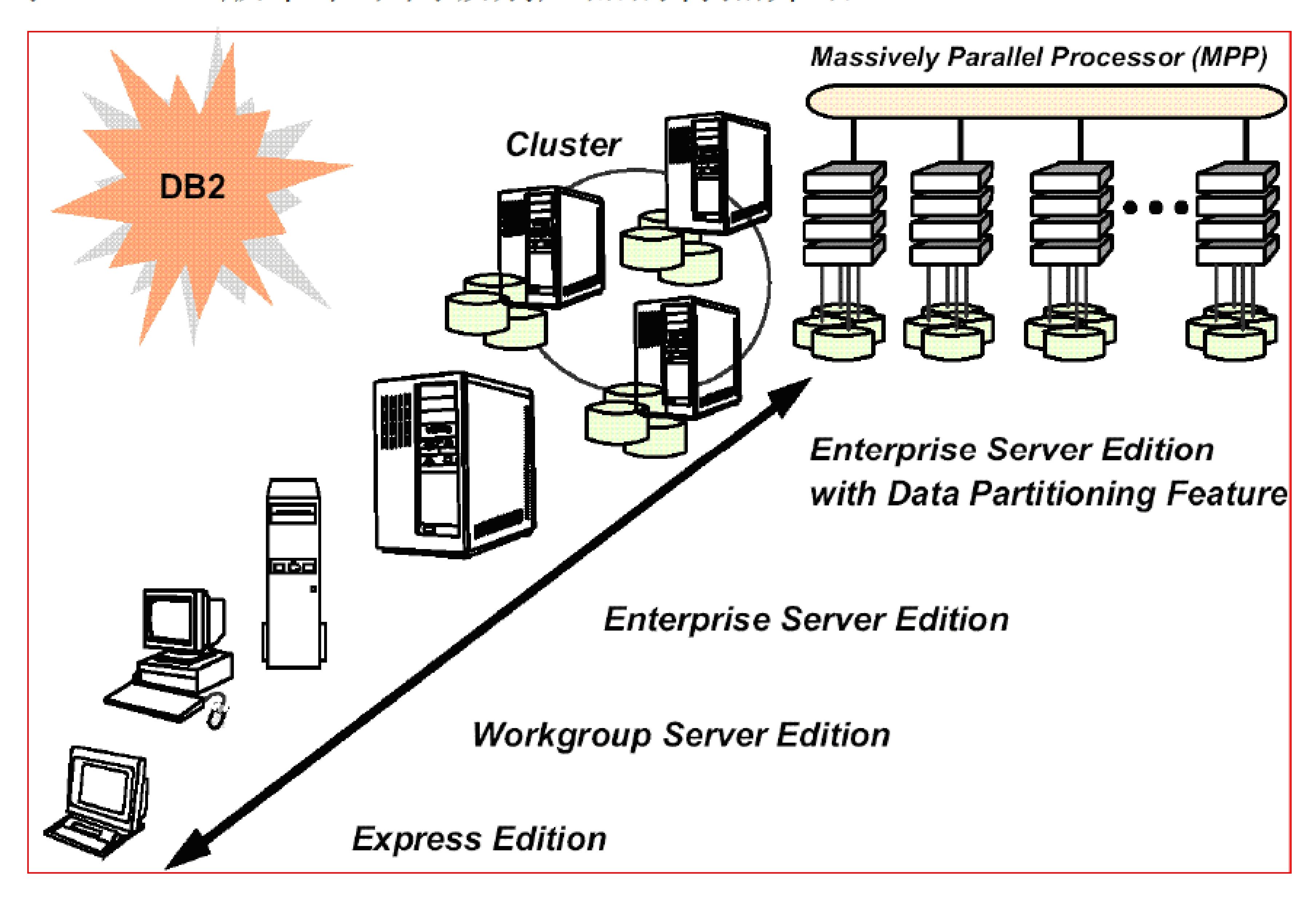


图 1-3　DB2 版本分类

DB2 V9 为各种类型的业务提供了正确的数据管理解决方案。提供包装了众多特性和功能的不同产品版本，以适应大量来自客户的不同需求。小型企业可以选择 Express Edition，中型企业可以选择 Workgroup，而 Enterprise Edition 则适合大型企业。除了这些版本，DB2

V9 另外提供了两个版本：Personal Edition 和面向开发的 Developer Edition，以及一个免费版 DB2 Express-C。表 1-1 描述了 DB2 V9 可用的版本。

表 1-1　DB2 V9 版本和平台支持

DB2 V9 分发版本
DB2 Express Edition V9 for Linux、UNIX and Windows DB2 Express V9 是一个功能完备的 DB2 数据服务器，它为中小企业(Small and Medium Business，SMB)市场提供了极具吸引力的入门级价格。该版本提供了经简化的程序包，可在应用程序内轻松进行透明安装。DB2 Express V9 可以轻松升级到 DB2 V9 的其他版本，它还具有和其他可伸缩性更高的版本相同的自主管理特性。DB2 Express V9 合并了本地 XML 数据存储，并允许使用 XQuery、XPath、SQL 和标准报告生成工具来灵活地访问 XML 数据
DB2 Workgroup Server Edition V9 for Linux、UNIX and Windows DB2 工作组服务器版用来满足数据服务器部署工作组或中型业务环境的需要。DB2 工作组服务器版包含了本地 XML 数据存储，并允许使用 XQuery、XPath、SQL 和标准报告生成工具来灵活地访问 XML 数据
DB2 Enterprise Server Edition (ESE) V9 for Linux、UNIX and Windows DB2 企业服务器版用来满足数据服务器处理大中型业务的需要。可以将它部署在任意大小(从一个 CPU 到任意数目的 CPU)的 Linux、UNIX 或 Windows 服务器上。DB2 企业服务器版是用于构建随需应变的企业级解决方案的理想平台，这些解决方案的示例如下：大小为 TB 级的大型数据仓库，高性能(每周 7 天，每天 24 小时全天候运行)的高容量事务处理业务解决方案，或者是基于 Web 的解决方案。DB2 企业服务器版包含本地 XML 数据，并允许使用 XQuery、XPath、SQL 和标准报告生成工具来灵活地访问 XML 数据。DB2 企业服务器版具有可选功能部件，用来在诸如数据库分区、性能、安全性、数据联合以及数据库管理方面提供附加的高级产品功能。此外，DB2 ESE V9 还提供了与其他 Enterprise DB2 和 Informix 数据源的连通性、兼容性以及集成
DB2 Enterprise Server Edition (ESE) V9 for Linux、UNIX and Windows With DPF DB2 ESE V9 With DPF 可以构建分区数据库，可以构建基于 MPP 的集群结构，主要应用于高性能计算领域，例如：数据仓库、科学计算、人工智能等
DB2 Personal Edition for Linux、UNIX and Windows DB2 Personal V9 是一个单用户、功能完备、具有内置复制的关系型数据库。对于基于桌面和膝上型电脑的部署是一个理想选择。DB2 Personal V9 可以进行远程管理，这使其成为在不要求多用户能力的不定期连接或远程办公实现中的最佳部署选择
Database Enterprise Developer Edition 此版本为单一应用程序开发人员提供软件包，用于设计、构建和原型化应用程序，以在任意 IBM 信息管理客户端或服务器平台上部署。可以面向 DB2 所有平台的开发。用于 DB2 的一组应用程序驱动程序包括：嵌入式 SQL、ODBC/CLI、JDBC/SQLJ、OLEDB、.NET、PHP、Perl 和 Ruby。数据访问和管理工具提供了 DB2 控制中心(Windows 和 Linux)和 DB2 命令行处理器(CLP)

(续表)

DB2 V9 分发版本
DB2 Express-C DB2 Express-C 是为社区提供的 DB2 Express Edition(DB2 Express)的一个版本。DB2 Express-C 是一个免费的数据服务器，可用于开发和部署 XML、C/C++、Java、.NET 和 PHP 应用程序。DB2 Express-C 最多可运行在双核 CPU、4 GB 内存的服务器上，以及对数据库规模或其他人为限制没有要求的任何存储系统

关于这些版本的详细许可协议超出了本书讨论的范围，但是需要注意每个版本的功能特性是不一样的。某些实用程序和功能仅在特定 DB2 数据库产品版本中可用。在某些情况下，实用程序或功能与特定 DB2 功能部件相关联，如果 DB2 Express 或 DB2 Workgroup 中没有免费包含某一功能，那么(在大多数情况下)可以通过附加的功能部件(Feature Pack)购买这一功能。

注意：

DB2 的分发版本就像我们使用 Windows XP 操作系统一样，有 Windows Home Edition，Windows XP Professional Edition 和 Windows XP 64-Bit Edition 等版本，每个版本包含的功能不一样，用户可根据自己需求来决定使用什么版本。

1.1.3 DB2 产品组件和功能

DB2 数据库的产品组件如图 1-4 所示。

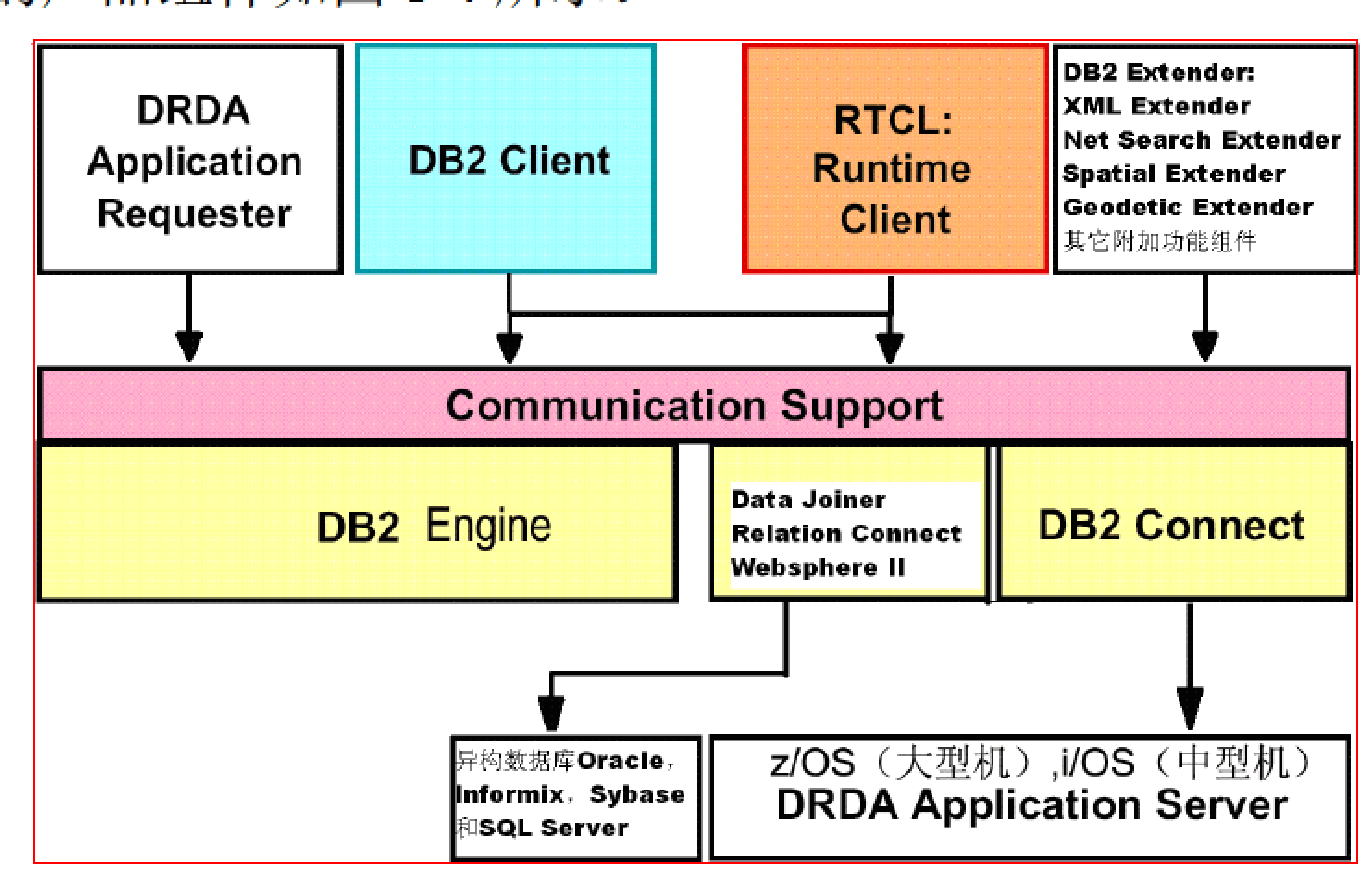


图 1-4　DB2 产品组件

DB2 Engine

DB2 Engine 是整个数据库系统的核心，提供了 DB2 的基本功能。DB2 引擎就类似汽车的发动机。它负责管理和控制对数据的存取；负责生成程序包(存储存取计划的数据库对象)；提供事务的管理；保障数据的完整性和数据保护；提供应用程序并发控制。数据库引擎(DB2 Engine)设计的完善与否，决定了数据库系统是否稳定和高效。DB2 Engine 是所有数据库中最强大的数据库引擎。

DB2 客户机

DB2 V9 大大简化了将应用程序连接到 DB2 数据库所需的基础设施的部署。DB2 V9 提供以下客户机：

- DB2 V9 Runtime Client (DB2 RTCL)

如果只需要让应用程序能够访问 DB2 V9 数据服务器，那么这就是最佳选择。它们提供了执行此任务所需的 API，但是这种客户机没有提供管理工具。DB2 运行时客户机(DB2 Runtime Client)的前身为 CAE(Client Application Enabler)，是所有 DB2 产品所共用的部件。它允许远程应用程序对数据库服务器进行存取。在这个组件中包含了 CLP(Command Line Processor)，允许用户动态地执行 SQL 语句和 DB2 命令，对本地和远程的数据库服务器进行存取。另外 Runtime Client 还提供了对 ODBC 和 JDBC 的支持，允许用户开发的 ODBC 或者 JDBC 应用程序对数据库进行存取。要想对数据库服务器进行存取，Runtime Client 几乎是必不可少的(在 WWW 上通过 Java Applet 存取是唯一的例外)。

DB2 运行时客户机为运行各种平台的工作站提供了访问 DB2 数据库的能力。它只提供基本连通性——不多不少刚刚好。如果您要建立到远程 DB2 服务器或 DB2 连接网关(Connect Gateway，它帮助您访问类似 DB2for z/OS 的大型机或主机系统上的 DB2)的连通性，那么您至少必须使用这个客户机。当然，您可以使用任何客户机进行连接。Runtime Client 的安装取决于操作系统，比如，如果需要在 Windows 操作系统上对数据库管理器进行存取，就需要在 Windows 系统上安装 Runtime Client for windows。

- DB2 V9 Client

包含 DB2 Runtime Client 中的所有功能，还增加了通过一组图形化工具进行客户机-服务器配置、数据库管理和应用程序开发的功能。DB2 V9 Client 整合了 DB2 V8 Application Development 和 DB2 V8 Administration Client 中的功能(DB2 管理客户机 Administration Client 是客户端的管理工具，它包含了一系列的图形化工具，用户可以方便地通过这些工具对数据库服务器进行远程管理。关于这些图形化的管理工具，我们在第 4 章会有比较详细的介绍)。Administration Client 的安装也取决于操作系统。这类客户机为各种平台的工作站提供了通过控制中心或配置助手(Configuration Assistant)访问并管理 DB2 数据库的能力。DB2 管理客户机具有 DB2 运行时客户机的所有功能，并且还包含所有 DB2 管理工具、文

档以及对瘦客户机的支持。DB2 应用程序开发客户机(DB2 Application Development Client)是专门为应用程序开发人员提供的，它包含了开发数据库应用程序所需要的各种组件，包括预编译器、Runtime Client、include 文件、库函数、样例程序和帮助文档等。

● Java Common Client(JCC)

这是一个 2 MB 的可重新发布的客户机，它提供了对 DB2 数据服务器的 JDBC 和 SQLJ 应用程序访问，而不需要安装和维护 DB2 客户机代码。如果要连接 DB2for System i 或 DB2 for System z 数据服务器，那么仍然需要安装 DB2 Connect 产品。

● DB2 V9 Client Lite

这个客户机是 DB2 V9 中新增的，它执行与 JCC 客户机相似的功能，但不是支持对 DB2 数据服务器进行基于 Java 的访问，而是用于 CLI/ODBC 应用程序。这个客户机尤其适合于那些希望将连接功能嵌入应用程序，而不需要重新发布和维护 DB2 客户机代码的 ISV。

通信支持(Communication Support)

通信支持提供了远程客户机支持，客户端应用程序可以通过多种网络协议对数据库服务器进行存取，TCP/IP、SNA(APPC/APPN)、NETBIOS 和 Name Pipe 等协议都可以被 DB2 所支持。

DB2 Relational Connect，DB2 Data Joiner

DB2 Relational Connect 和 DB2 Data Joiner 通过允许用户和应用程序存取存储在 Oracle、Sybase、Informix、SQL Server 等数据库中的数据，增强了 DB2 数据库中所包括的分布式请求功能。它们允许将驻留在多个不同平台上的数据作为单个数据库映像访问，这些数据既可以是 IBM 的，也可以是其他供应商的；既可以是关系型的，也可以是非关系的。这可以与内置分布式查询功能一起使用，以使 DB2、Oracle、Sybase、Informix、SQL Server 等数据库之间的查询 SQL 化。这些组件可以帮助企业整合数据，以加速 Oracle、Sybase、Informix、SQL Server 等数据库源中的选择性能，以便进行数据集中。

这两个功能今天已经被整合成 IBM 的一个产品——WebSphere Information Integrator。

DB2 Connect

许多大型组织中的大量数据由 DB2 for i5/OS、DB2 for z/OS 等数据服务器进行管理。有了 DB2 Connect 的帮助，在任何支持的 DB2 分布式平台上运行的应用程序都可以透明地操作这些数据，就像是本地数据服务器在管理数据一样。还可以将 DB2 Connect 及其相关工具与许多现成的或定制开发的数据库应用程序一起使用。DB2 Connect 提供了从 Windows、Linux 和 UNIX 开放平台连接大型机和中型机的能力。

DB2 扩展器(Extender)

DB2 扩展器使数据库应用程序能够超越传统的数字和字符数据，为底层数据服务器提供额外的功能。这部分组件是可选的，包括：DB2 XML Extender 扩展器(处理 XML)；DB2 Net Search Extender 扩展器可帮助企业在搜索数据库中的数据时获得更高的性能；DB2 Spatial Extender 扩展器可以在 DB2 中与文本和数字等传统数据一起存储、管理和分析空间数据——关于地理特征位置的信息；DB2 Geodetic Extender 扩展器可以将地面作为球体对待，从而消除投影等操作造成的不精确。

注意：
大家了解这些可选的扩展器即可，通常情况下用不到这些组件。

通过上面的讲解我们大概了解了 DB2 数据库的产品组件和功能，其实有些组件我们很少用到，就像 DB2 Connect，如果您单位没有大型机或中型机，那么您可能不会用到它。

1.2　DB2 数据库安装配置

在我们了解完 DB2 的相关版本和产品组件后，将开始我们的 DB2 学习之旅。在学习之前我们要首先安装 DB2 数据库，下面我们讲解如何在不同的平台上安装 DB2，这部分内容相对简单，为了节省篇幅，我们把部分图形化界面省去了。

DB2 安装方法

表 1-2 列出了不同操作系统上 DB2 的安装方法。

表 1-2　不同操作系统可用的安装方法

安 装 方 法	Windows	Linux 或 UNIX
“DB2 安装”向导(图形化界面)	是(setup.exe)	是(db2setup)
响应文件安装(静默安装)	是	是
db2_install 命令(命令行界面)	否	是(db2_install)

“DB2 安装”向导(db2setup)

“DB2 安装”向导是可在 Linux、UNIX 和 Windows 操作系统上使用的一个 GUI 安装程序。“DB2 安装”向导提供了易于使用的界面，用于安装 DB2 产品和执行初始设置与配

置任务。Linux/UNIX 上启动时需要配置 JRE 运行环境和 X 环境。

“DB2 安装”向导还可以用来创建 DB2 实例和响应文件，它们可用于在其他机器上复制此安装。这种安装方法比较简单，一般用得最多。

响应文件安装

响应文件是一个包含设置和配置值的文本文件。DB2 安装程序将读取该文件，并根据已指定的值来执行安装。响应文件安装也称为静默安装。响应文件的另一个优点是：它们提供了对那些不能使用“DB2 安装”向导设置的参数的访问。在 Linux 和 UNIX 操作系统上，如果将 DB2 安装映像嵌入您自己的应用程序中，那么您的应用程序有可能从安装程序中以计算机可读的格式接收安装进度信息和提示。这种安装方式一般比较适合大批量的客户端安装。

db2_install 命令(仅适用于 Linux 和 UNIX 平台)

db2_install 命令将安装您指定的具有“字符”界面支持的 DB2 产品的所有组件。通过使用-L 参数就可以选择要支持的其他语言。尽管 db2_install 命令会安装您指定的 DB2 产品的所有组件，但它不会执行用户和组创建、实例创建或配置。在安装之后执行配置时，此安装方法可能是首选。如果您希望在安装 DB2 产品时配置它，那么请考虑使用“DB2 安装”向导。

1.2.1 DB2 在 Windows 上的安装

在 Windows 上安装 DB2 非常简单，安装步骤如下：

首先检查系统硬件资源满足安装的最小需求，然后启动 DB2 安装向导(Windows)。

(1) 用管理员账号登录 Windows 系统。

(2) 关闭所有应用程序，以便于安装 DB2 可以快速完成和安装过程出现问题时易于定位。

(3) 把 DB2 的安装光盘插入光驱，启动自动安装向导，如图 1-5 所示。

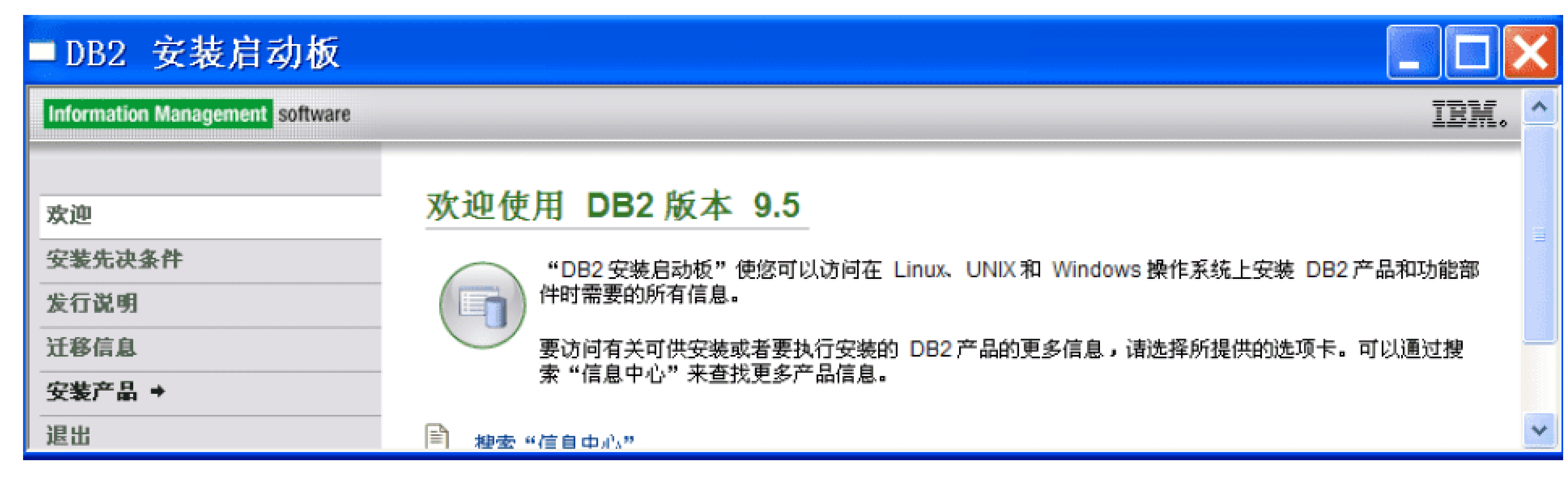


图 1-5　启动 DB2 安装启动板

在 DB2 安装向导的启动界面中，可以看到左边菜单有“安装先决条件”、“发行说明”、“迁移信息”、“安装产品”和“退出”这几个选项。可以预先浏览“安装先决条件”，以了解安装 DB2 有哪些具体要求和注意事项。这里需要强调的是，若安装选择的组件所需的空间超过了为安装这些组件而指定的路径所在磁盘的空间，安装程序会发出关于空间不足的警告。可以继续安装。但是，如果实际上没有足够的空间用于正要安装的文件，当没有更多的空间时，安装将停止。此时，如果不能释放空间，将必须人工停止安装程序。

浏览“发行说明”，就可以了解到与 DB2 有关的信息指南。初次接触 DB2 通用数据库 V9 的用户最好访问一下“快速导览”，这有利于从整体上把握 DB2 通用数据库的梗概。

(4) 单击“安装产品”。

(5) DB2 安装向导。

DB2 安装向导检查系统是否满足所有系统需求，以及目前是否存在任何 DB2 安装，如图 1-6 所示。单击“下一步”按钮继续安装。



图 1-6　检查是否满足所有系统需求

(6) 许可协议。

阅读并接受许可协议(选择“我接受...”单选按钮)，如图 1-7 所示。单击“下一步”按钮继续。

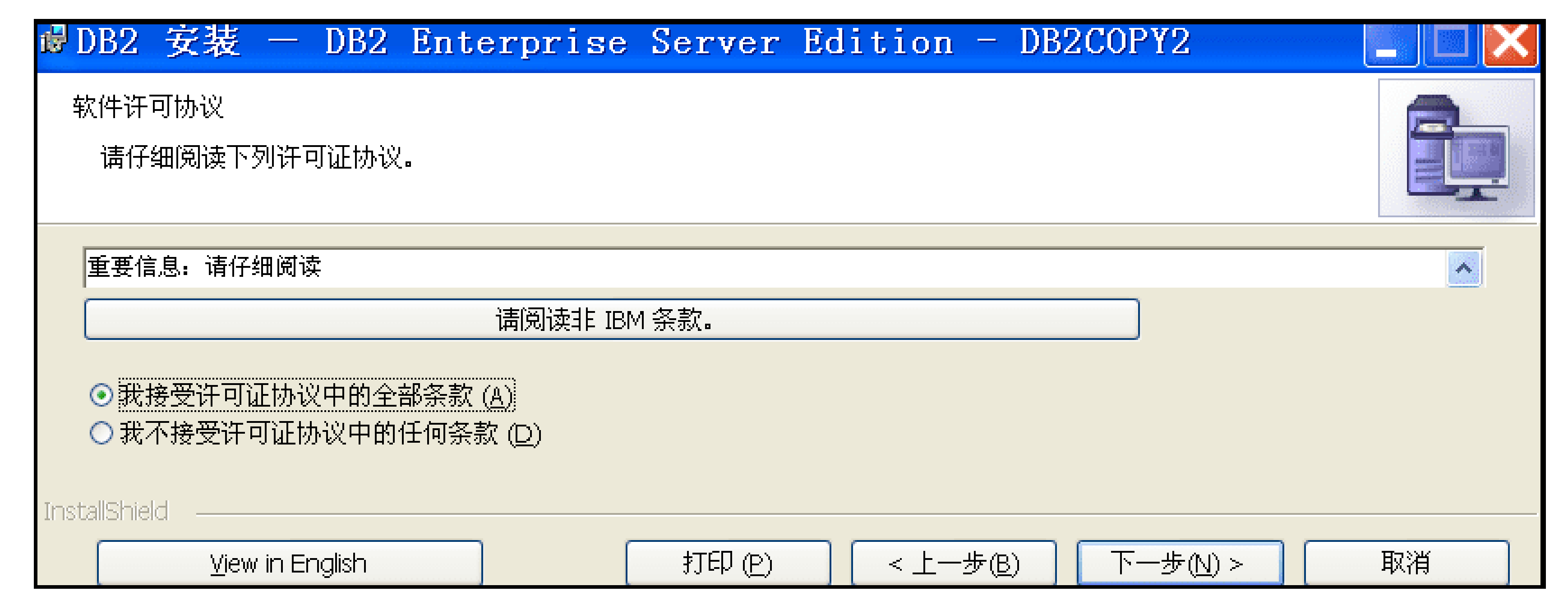


图 1-7　阅读并接受许可协议

(7) 选择安装类型。

对于本书，选择“典型安装”选项(这是默认设置)，如图 1-8 所示。“精简安装”选项执行基本安装，而“定制安装”选项允许用户定制希望安装的特性。单击“下一步”按钮继续。

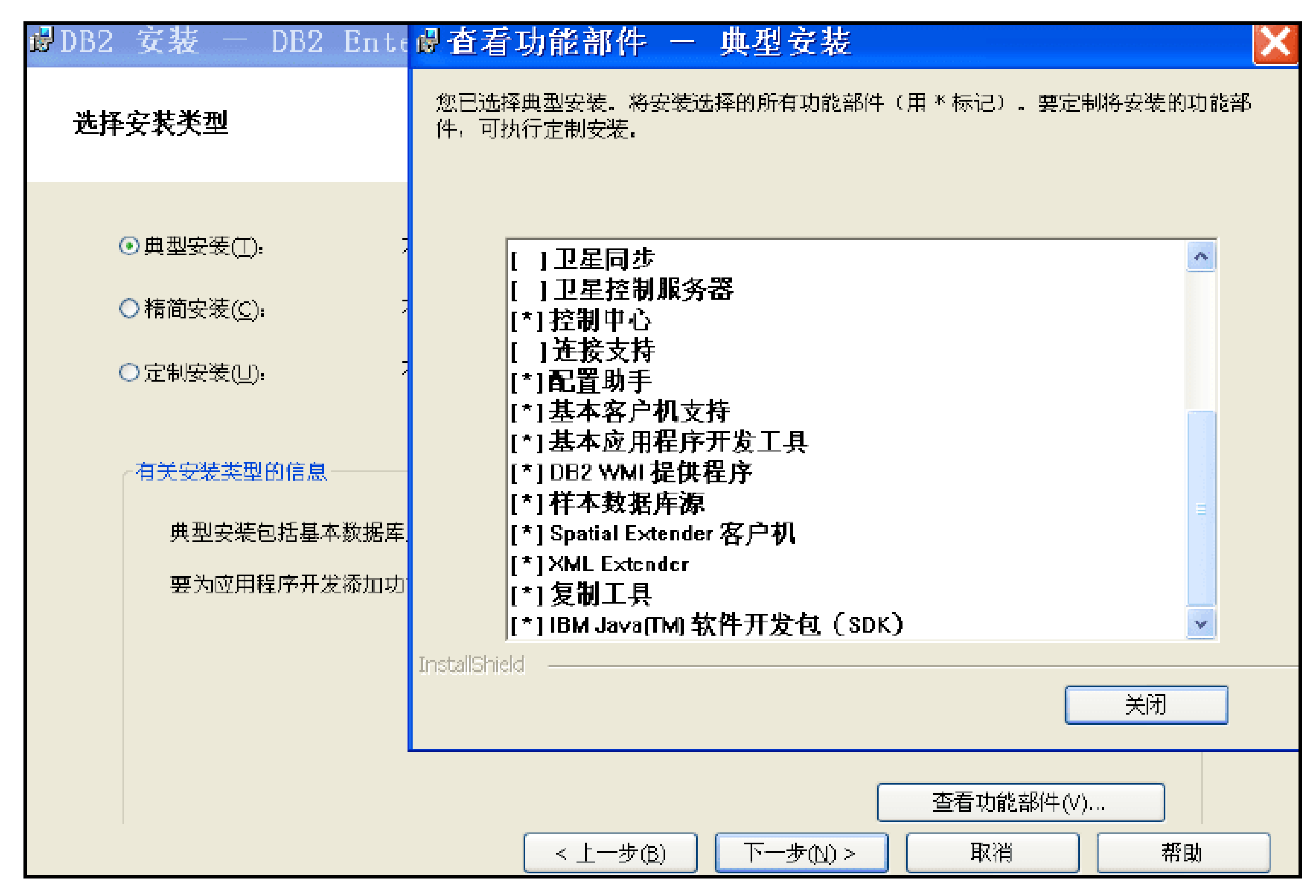


图 1-8　选择安装类型

可以选择是否创建响应文件以便以后执行响应文件安装，如图 1-9 所示。

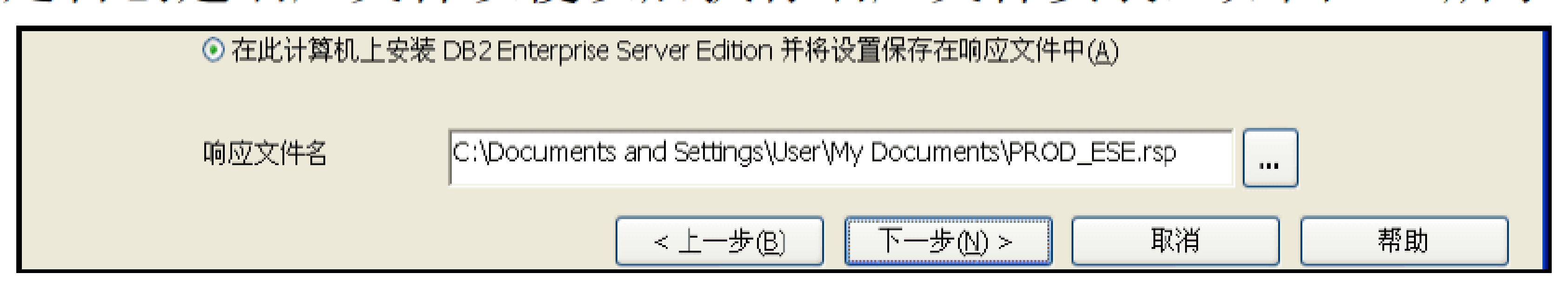


图 1-9　选择创建响应文件

(8) 选择安装文件夹。

图 1-10 允许您选择安装 DB2 代码的驱动器和目录。要确保安装位置有足够的空间。对于这个示例，使用默认的驱动器和目录设置(如下所示)：

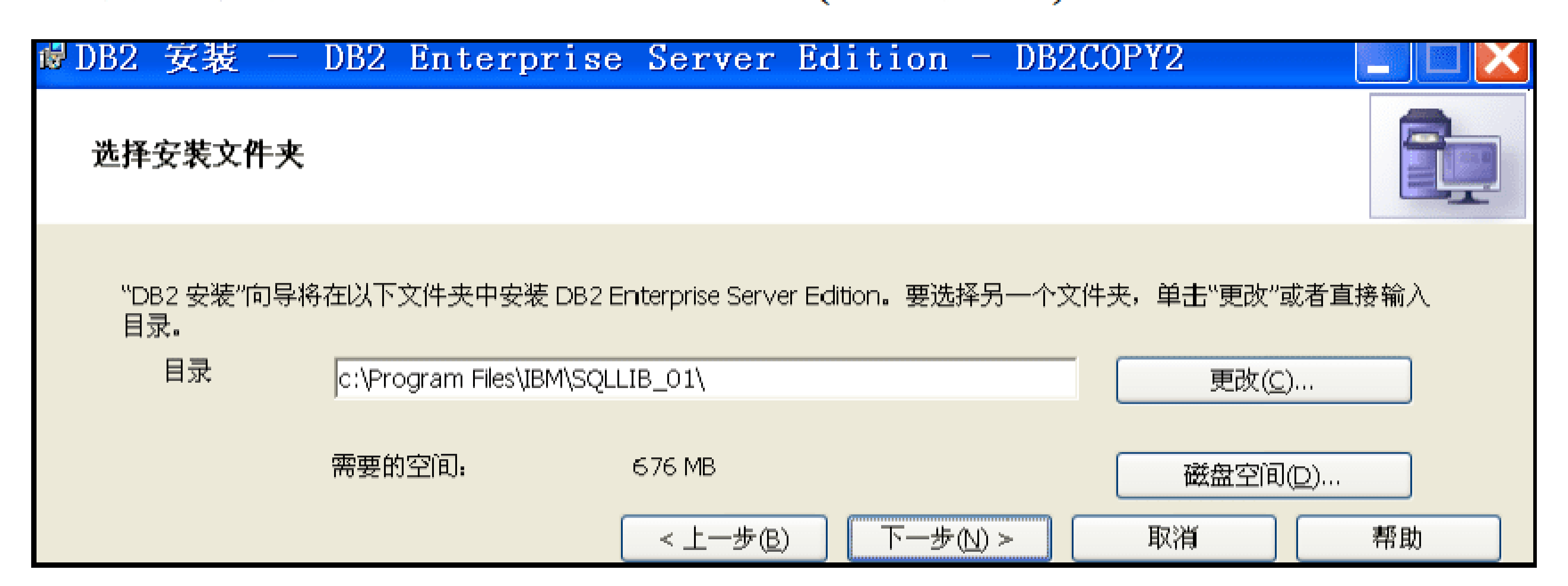


图 1-10　选择安装文件夹

驱动器：C:

目录：C:\Program Files\IBM\SQLLIB_01

单击“下一步”按钮继续。

(9) 配置 DB2 实例。

可以认为 DB2 实例是数据库的容器。必须有一个实例，然后才能创建数据库。在 Windows 上进行安装时，会自动创建一个称为 DB2 的实例。后面第 2 章将详细讨论实例。

在默认情况下，DB2 实例监听端口 50000 上的 TCP/IP 连接，如图 1-11 所示。通过单击“协议”和“启动”按钮，可以分别修改默认的协议和端口。在这个示例中，建议使用默认设置。单击“下一步”按钮继续。

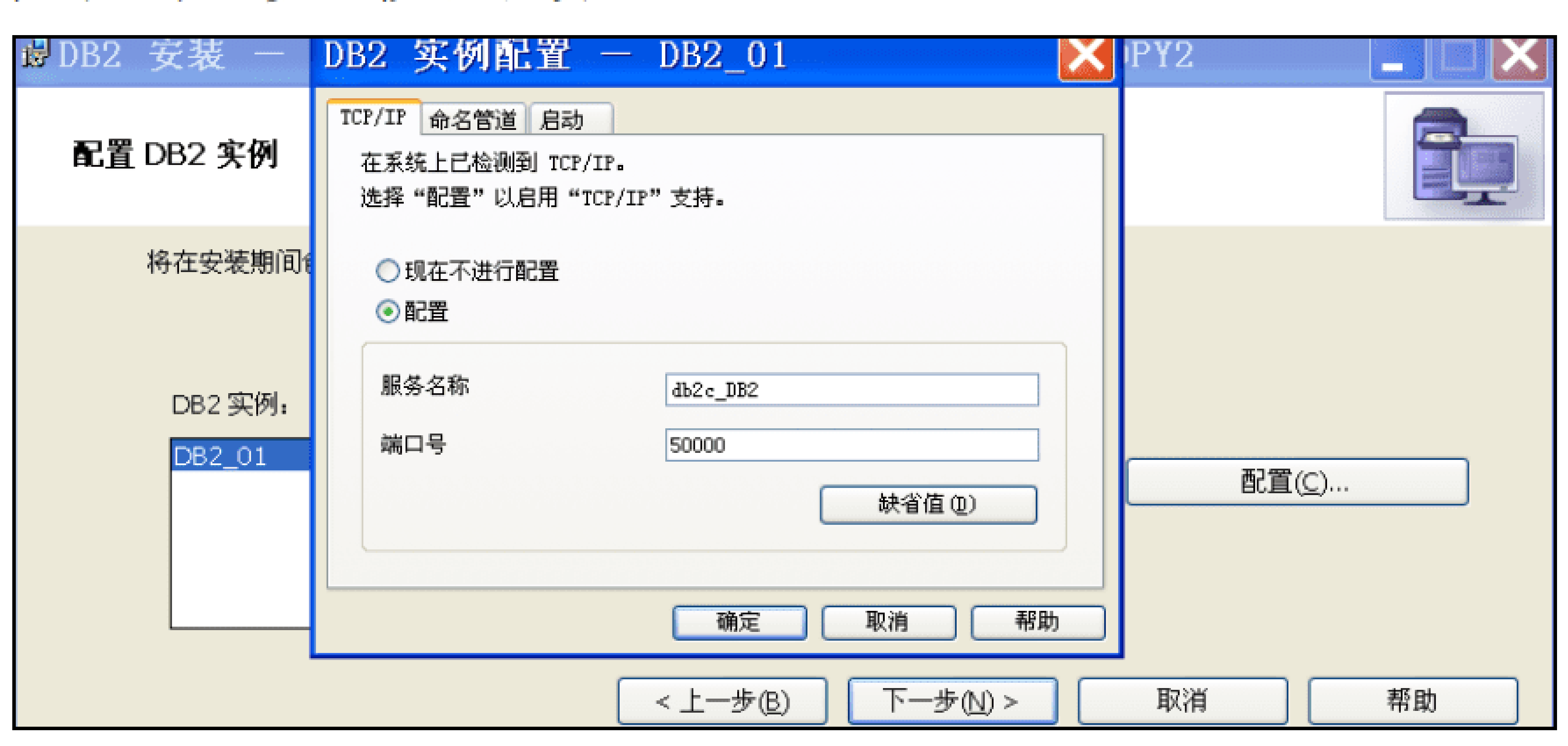


图 1-11　配置 DB2 实例

(10) 设置用户信息。

安装 DB2 之后，某些 DB2 进程会作为系统服务运行。为了运行这些服务，需要一个操作系统账户。在 Windows 环境中，建议使用默认的 db2admin 用户账户，如图 1-12 所示。如果这个用户账户不存在，DB2 会在操作系统中创建它(所以安装时需要管理员账号)。也可以指定使用一个现有的账户，但是这个账户必须具有本地管理员权限。在本书中，我们建议使用 ibmdb2 作为密码。单击“下一步”按钮继续。此时创建的用户 db2admin 具有最高权限，在 Windows 系统中 DB2 服务的启动都是以该用户的权限在执行。同时，该用户所属的组(默认为 db2admins)将作为 DB2 实例的最高管理权限组，拥有新创建实例的所有权限，如创建数据库、删除数据库、备份数据库等。属于该组的所有用户都可以执行这些动作，在默认安装下，该组只有 db2admin 一个用户。我们在后面的 DB2 数据库安全部分会详细解释这些。

(11) 启用操作系统安全性。

指定是否想对计算机上的 DB2 文件、文件夹、注册变量和其他对象启用操作系统安全性，如图 1-13 所示。



图 1-12　设置用户信息



图 1-13　启动操作系统安全性

(12) 开始安装

在经过前面一系列的选择后，我们开始安装 DB2 软件，如图 1-14 所示。

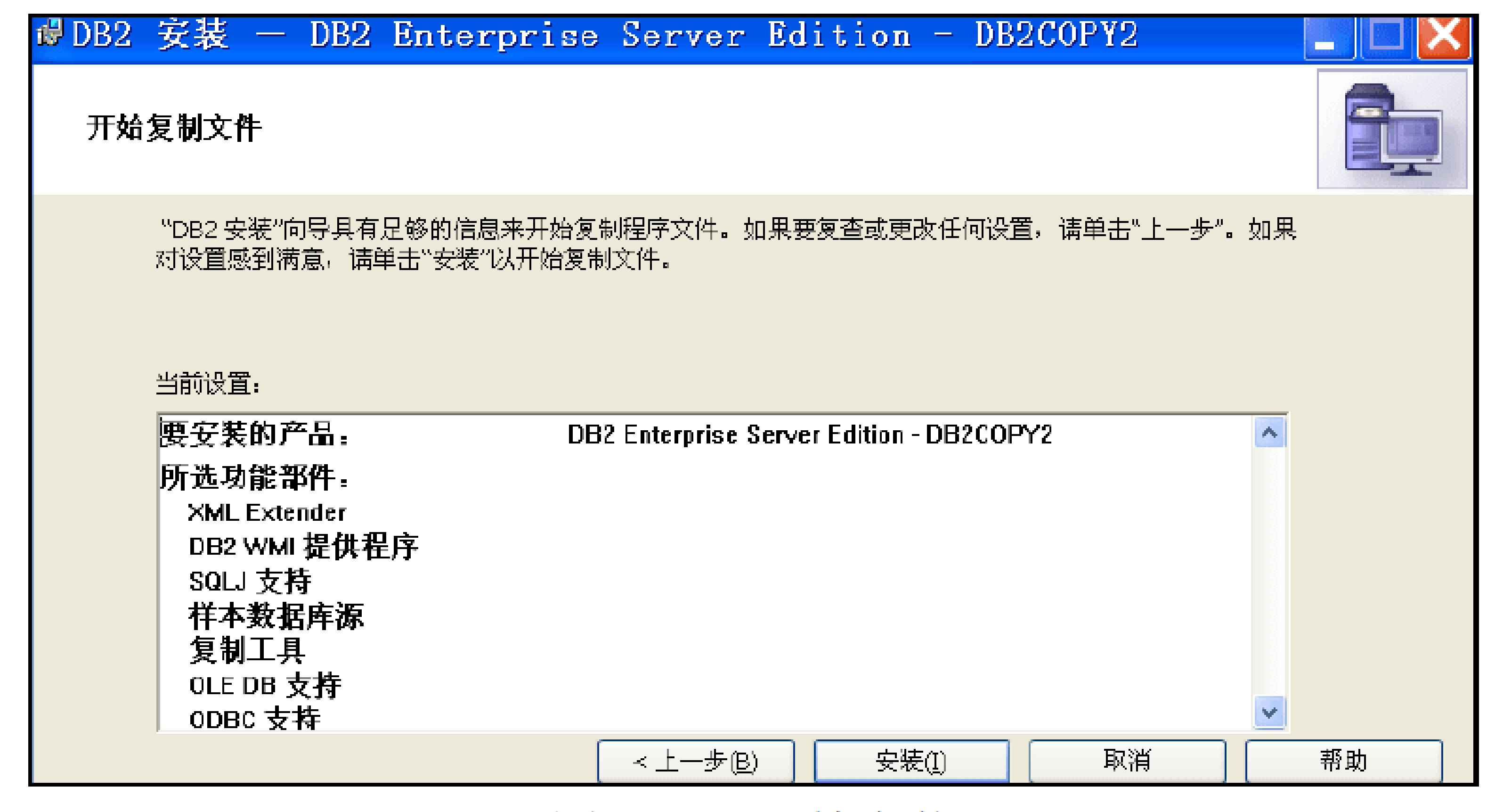


图 1-14　开始安装

(13) 完成安装，如图 1-15 所示。



图 1-15　完成安装

(14) 验证安装。

在安装完成之后，显示一个称为“DB2 第一步”的启动面板(也可以用命令 db2fs 启动它)。

在“DB2 第一步”启动面板中，选择“创建数据库”选项卡，然后按照向导的指示创建 SAMPLE 数据库，如图 1-16 所示。

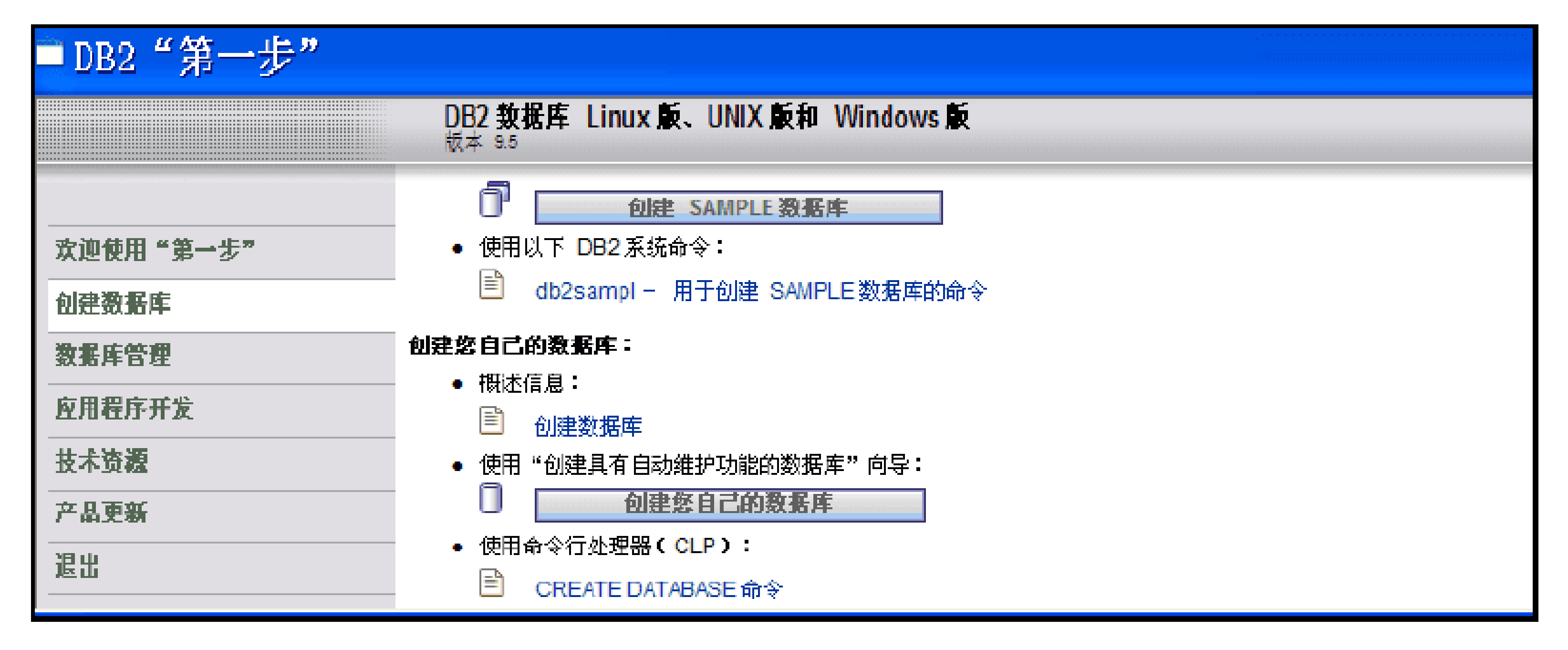


图 1-16　选择并创建 SAMPLE 数据库

选择“XML 和 SQL 对象和数据”选项并单击“确定”按钮，如图 1-17 所示。

(15) 创建数据库进行过程中。

在创建数据库时，显示下面的进度屏幕(这个过程可能要花几分钟)，如图 1-18 所示。

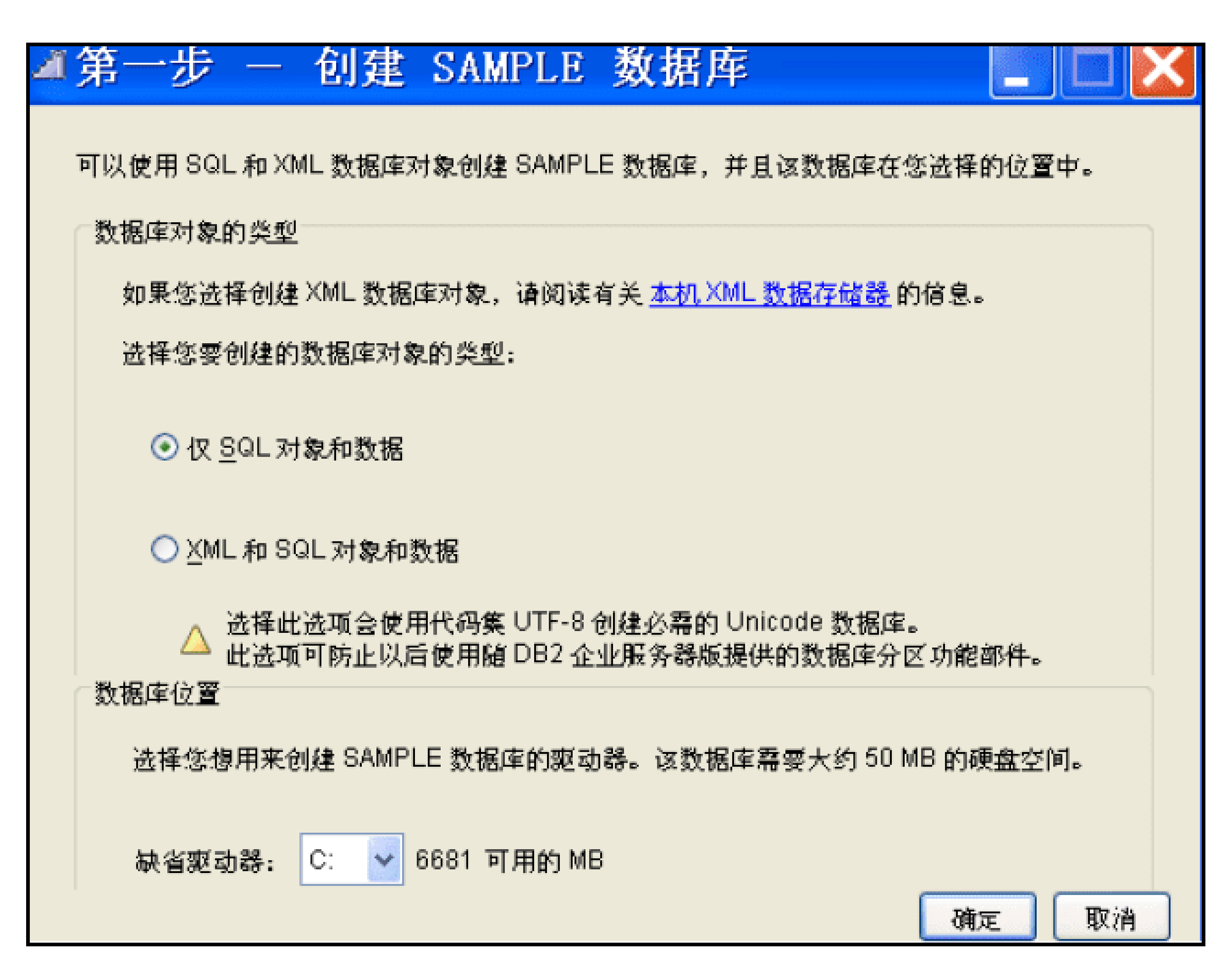


图 1-17　选择“XML 和 SQL 对象和数据”选项



图 1-18　正在创建数据库

(16) 完成创建数据库。

在数据库创建过程完成后，如图 1-19 所示，单击“数据库管理”按钮，并找到“控制中心”按钮，单击打开。



图 1-19　完成创建数据库

打开后，并检查左边面板中现在是否出现了名为 SAMPLE 的数据库，如图 1-20 所示。可能必须刷新控制中心视图，才能看到新的变更。注意这个 SAMPLE 数据库，我们本书的所有练习和实验都是在这个数据库上实现的，所以建议初学者创建它。

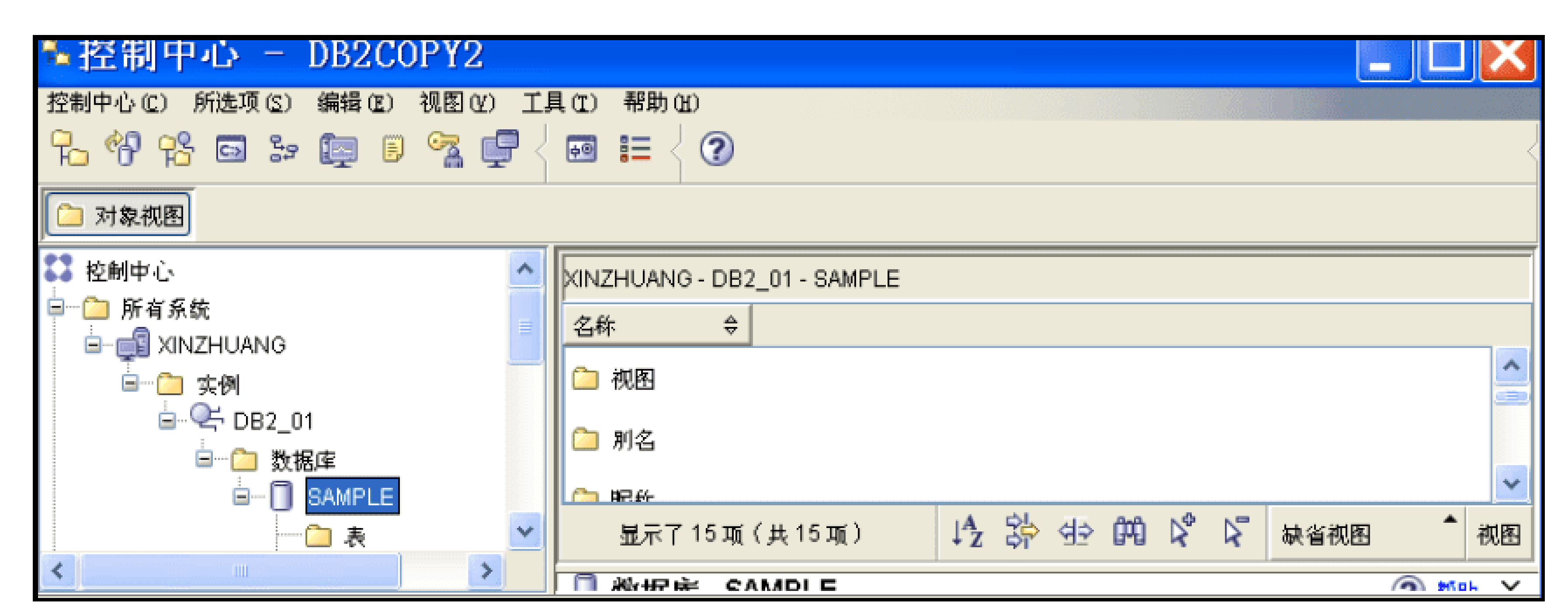


图 1-20　确认已成功创建数据库 SAMPLE

(17) 重新启动计算机，完成安装。

尽管正式的 DB2 安装文档中没有提到这个步骤，但是我们建议重新启动系统(如果可能的话，不重启也是可以的)，从而确保成功地启动所有进程并清理内存资源。这个步骤是可选的。

1.2.2　DB2 在 Linux/UNIX 上的安装

DB2 在 AIX、HP-UX、Sun Solaris 以及 RedHat Linux 上的安装步骤基本类似，只是在安装前的准备工作上有差异。在这些平台上如果调用 db2setup 安装向导，安装界面和 Windows 的图形界面基本上差不多，您需要配置好 JRE 运行环境和 X 环境。

我们建议大家使用 db2_install 命令行安装 DB2。而且我建议读者学习的时候最好能在本地机器上利用 VMware 虚拟机安装 Linux 操作系统，然后在 Linux 上安装 DB2 数据库并在这个环境中学习和做实验。因为通常情况下大多数比较重要的 DB2 数据库都是运行在 UNIX 环境中的，而 Linux 是最接近 UNIX 的环境。

那么如果要在 Linux/UNIX 上安装 DB2 数据库，一般需要注意以下几点：

- 使用 root 用户安装(DB2 V9 可以使用非 root 用户安装，但是有一些限制，建议读者还是使用 root 用户安装)。
- 确保硬件平台、内存和硬盘满足安装的最低要求。
- 确保正确地设置安装所需的内核参数。
- 确保操作系统版本补丁、操作系统内文件包和 DB2 的版本相兼容。

在正确设置好以上必须的环境和参数后，下面我们使用 db2_install 来安装 DB2。

安装步骤：

db2_install 脚本会安装 DB2 产品中您指定的所有组件，并具有字符界面支持。它并不执行用户和组创建、实例创建或配置。按照以下步骤开始使用 db2_install 进行安装：

以 root 用户登录。插入包含 DB2 软件的介质 DVD，或者访问存储安装映像的文件系统。改变目录至 <DVD_mount>/ese/disk1。

运行以下命令：

```
#./db2_install -b DB2DIR -p productName
```

其中，DB2DIR 是要安装 DB2 产品的路径，productName 是要安装的产品的名称。对于完整的 Enterprise Server Edition，选择 ESE；也可以选择客户机(CLIENT)或运行时客户机版本(RTCL)。

可以不为 db2_install 命令提供任何参数，在这种情况下会提示您输入产品名称和安装路径。可以通过运行命令 db2_install –h 了解详细的用法信息。如果没有使用－1 选项指定日志路径，那么可以在/tmp 目录中找到安装日志文件。

安装界面：

```
<DVD_mount>/db2install/V9.5/ese/disk1 #./db2_install
Default directory for installation of products - /opt/ibm/db2/V9.5
***********************************************************
Do you want to choose a different directory to install [yes/no]
yes
Enter full path name for the install directory -
------------------------------------------------
/opt/ibm/db2/V9.5
Specify one or more of the following keywords,
separated by spaces, to installDB2products.
  CLIENT-----------------------客户端
  RTCL--------------------------运行期客户端
  ESE------------------------------ESE 服务器
Enter "help" to redisplay product names.
Enter "quit" to exit.
***********************************************************
ESE
DB2 installation is being initialized.
Total number of tasks to be performed: 39
Total estimated time for all tasks to be performed: 853
```

在 Linux/UNIX 上安装后，DB2 的安装目录如图 1-21 所示。

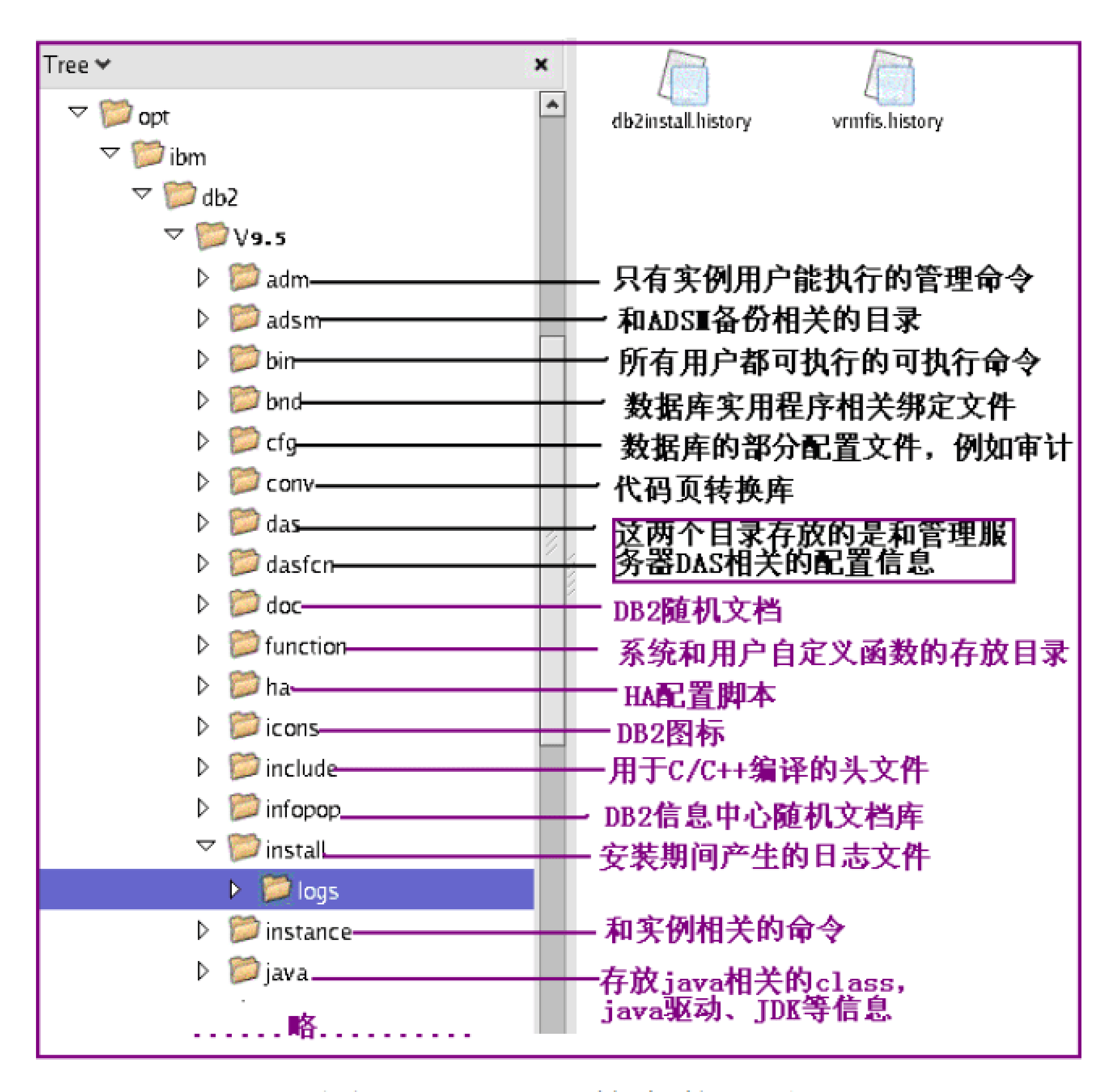


图 1-21　DB2 的安装目录

1.3　DB2 数据库体系结构

安装完 DB2 软件后，在我们开始学习 DB2 之前，先简单了解一下 DB2 数据库的体系结构。图 1-22 显示了 DB2 V9 数据库对象的体系结构。

系统

DB2 体系结构中的最高一层是系统，一个系统表示 DB2 的一个安装。在一个由很多机器组成的网络环境中，我们有时也称系统为数据库分区。一个系统可以包含多个 DB2 实例，每个实例能够管理一个或多个数据库。

An instance

实例也称为数据库管理器(Database Management Application)，是数据库管理器在内存中的映像，是管理数据的 DB2 代码。实例相当于 Informix 的 Informix Server，在一台机器上可以有多个相互独立的实例，实例之间是彼此独立，同时运行，不会相互影响。每个实例可以管理若干个数据库，一个数据库只属于一个实例。它可控制对数据执行的操作，并管理分配给它的系统资源。每一个实例都是一个独立的运行环境，在该环境中可以编目数据库和设置配置参数。可以在同一物理服务器上创建多个实例，并为每个实例提供唯一的

数据库服务器环境。

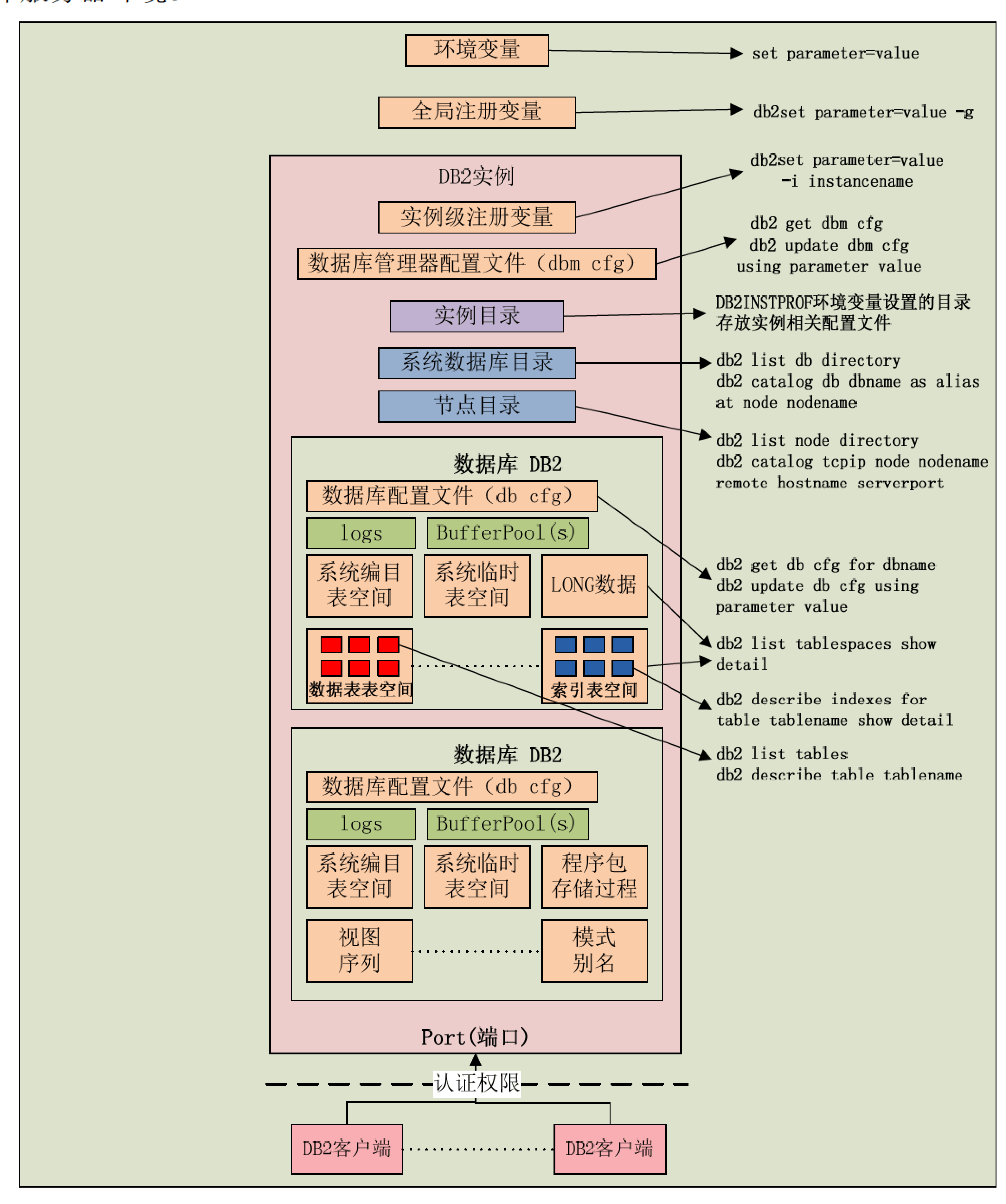


图 1-22　DB2 体系结构概览

关于如何创建实例，管理实例等详细内容，我们会在“第 2 章：创建实例和管理服务器”一章中详细讲解。

Configuration files and the DB2 profile registries

和许多其他关系数据库管理系统(RDBMS)一样，DB2 使用不同的配置参数来管理、监视和控制 DB2 系统的行为。这些机制包括：

环境变量

环境变量是在操作系统级别上定义的变量。例如，在 Windows 平台中，您可以为一个变量创建新的项，或者通过选择**“控制面板”**-->**“系统”**-->**“高级”**-->**“环境变量”**来编辑现有变量的值。在 UNIX 中，您通常可以将安装了 DB2 之后所提供的脚本 db2profile (Bourne 或 Korn shell)或 db2cshrc(C shell)添加到.login 或.profile UNIX 初始化文件中。db2profile/db2cshrc 文件包含了“export”UNIX 命令，这些命令能确保在数据库用户的环境中每次调用 shell 时都会传递 UNIX 环境变量的值。在创建实例后我们需要配置相关环境变量来为用户提供一个运行环境，这部分内容会在“第 2 章：创建实例和管理服务器”中讲解。

DB2 概要文件注册变量

DB2 通过使用概要文件注册变量来集中控制环境变量，不同的概要文件提供了不同级别的支持。注册变量定义了 DB2 操作环境，这些变量存储在 DB2 注册文件中，具有两层注册变量。一种是全局变量，变量的设置是系统范围的。另一种是实例变量，变量的设置用于特定的实例。在实例上定义的变量值能够覆盖全局变量中同名变量的设置。

在单分区数据库中，有 3 个概要文件注册变量：

- DB2 实例级概要文件注册变量：DB2 变量的大多数放在这个注册文件中。特定实例的变量设置放在这个注册文件中。可以在全局级(db2set -g 选项)和实例级(db2set -i 选项)设置 DB2 概要文件注册变量。实例层定义的值能够覆盖全局层中同名变量的设置。
- DB2 实例概要文件注册表：此注册表包含系统可识别的所有实例名的列表。可通过运 db2ilist 查看系统上提供的所有实例的完整列表。
- 环境变量：这类变量的设置方法因操作系统的不同而有所差异。

系统启动时，检查系统的变量时，**按照先环境变量，再全局级注册变量，最后实例级注册变量的顺序来搜索**。关于这部分变量的设置在第 2 章有详细讲解。

配置参数

配置参数是在两个不同的级别(实例级和数据库级)上定义的。每个级别上的变量都是不同的(不像注册表变量那样可以在不同级别上定义相同的变量)。

配置参数能够影响数据库或者数据库管理员的操作特性，它们存储在配置文件中。数据库管理器配置参数是在 DB2 实例创建的时候创建的，其中所包含的参数能够在实例层、独立于实例的任何数据库影响系统资源。

许多参数都会被数据库设置为默认值，管理员可以根据自己的需要修改它们。数据库的配置文件是在数据库创建的时候创建的，它位于数据库所在的目录。每个数据库有一个配置文件。数据库的配置参数定义了分配给数据库的各种资源的数量。许多的配置参数值可以被修改，以满足数据库运行的各种需求，如提高性能和容量等。不同参数的修改依赖于特定数据库应用的需求。

关于配置参数的阐述我们分别在“第 2 章：创建实例和管理服务器”和“第 3 章：创建数据库和表空间”中讲解。

Databases

关系数据库使用一组表来管理数据，一个表由在行和列中以逻辑关系排列的数据组成，每个表的数据在逻辑上相关，在表之间能够定义关系。

每个数据库包含一组系统编目表(或称之为数据字典)、配置文件和恢复日志，系统编目表用于描述数据的逻辑和物理结构，配置文件包含所有为数据库分配的配置参数值，恢复日志记录正在进行的事务处理和可存档的事务处理。

数据库可以是本地的，也可以是远程的。本地数据库物理上位于本地的机器上；当数据库物理上驻留在另一台机器上时，则称为远程的。关于数据库的详细讲解请参见“第 3 章：创建数据库和表空间”。

Tablespaces

表空间是数据库中表数据与数据库之间的逻辑中间层，数据库中的物理空间组织为表空间的集合，而表空间是表的逻辑集合。每个表空间包含容器集合，容器(container)是用来描述物理空间分配的一般术语。数据库将数据存储在自己的表空间容器中。

每个表存储在一个或几个表空间中，为了提高性能，或者为了便于表空间的备份，可以将一个表中不同类型的数据分别存储在不同的表空间中，如常规数据存储在第一个表空间中，将一个表的索引存储在第二个表空间中，将大对象数据存储在第三个表空间中。

表空间最终会映射到物理存储介质上，对物理存储的合理使用可以让管理员有效地控制数据库的性能。例如，可以使用最快的设备或内存硬盘用于存储频繁使用的表，使用较慢的设备存储不经常使用的数据。表空间的概念提供了对底层存储物理设备的更加灵活的使用。

表空间的规划设计会显著影响数据库运行的性能。关于表空间的详细规划设计，我们

会在“第 3 章：创建数据库和表空间”中详细讲解。

Connectivity and DB2 directories

● 节点目录

节点目录用于存储远程数据库的所有连通性信息。节点目录中记录与每个系统进行通信所需要的信息：例如机器(其中包含了您想连接的数据库)的主机名或 IP 地址，还有相关的 DB2 实例的端口号和使用的通信协议。要想得到您想要连接的远程实例的端口号，可以通过查看该实例的 dbm cfg 中的 svcename 参数来实现。该值通常对应于 TCP/IP services 文件中的某一项。节点目录主要用于配置客户端到服务器的通信使用。

● 系统数据库目录(或系统 db 目录)

系统数据库目录包含本地数据库目录和从远程的机器上映射到本地的数据库目录。它是我们访问数据库的一个入口，我们连接数据库时首先去系统数据库目录中判断这个数据库是否存在，然后再判断这个数据库是本地数据库还是远程数据库。如果是本地数据库就直接到本地物理目录上访问；如果是远程数据库，还要找到这个远程数据库在哪个节点上，然后再到节点目录中找到这个节点的通信信息，最后需要连接到远程的节点上来访问这个远程的数据库。

● 本地数据库目录(或本地 db 目录)

本地数据库目录包含了有关本地数据库(即驻留在您目前正在使用的机器上的数据库)的信息。当您使用 create database 命令创建数据库时，在该目录中会添加一个条目。

关于节点目录、系统数据库目录和本地数据库目录的详细信息，我们会在“第 4 章：访问数据库”中详细讲解。

模式

模式是数据库对象的逻辑分组集合，它细化了数据库的“粒度”，帮助对表和其他数据库对象进行逻辑分组。模式可以归个人所有，拥有者可以控制对数据以及其中的对象的存取。

模式是数据库对象特征划分的结果集，它可以表示数据库对象集的特点，有一定的安全作用。数据库中所建的每一个对象都有模式，这些模式会隐式或显示地增加为对象的前缀。创建用户时，系统会为每个用户默认隐含建立与用户名同名的模式名。当创建数据库中的对象时，如果写明了它的模式名(即对象的前缀)，则此模式即为该对象的模式；如果未指明模式名，那么与当前用户名同名的模式即为当前对象的模式。关于模式的创建我们会在“第 5 章：创建数据库对象”中讲解。

Tables, indexes and large objects

表是数据库的基本组成单元，是客观世界中实体的一种描述。表由行、列组成。表的每列描述了对应实体的一个属性，同一列的数据都具有相同的数据类型。表的每一行都描述了一个实体的信息。所有数据库和表数据都被存储在表空间中，表中的数据在逻辑上是相关的。可定义表之间的关系。索引是与表相关的有序指针集，用于性能目的并确保唯一性。视频、音频和扫描文档等可以作为大对象(LOB)存储在数据库中。表、索引和 LOB 驻留在表空间中。为了提高性能，可以把表数据、索引数据和大对象数据分别存放到不同的表空间，关于这部分的详细讲解，请参见“第 5 章：创建数据库对象”。

视图

视图是高效率的数据操纵机制。视图是“虚拟”的表，视图不是真正的表，不需要永久性存储器，它的数据本质上还是来自于数据库中的基表。

视图是从一个或几个基本表导出的表，也可从其他视图导出。某一用户可以定义一个或多个视图，经授权后一个视图也可为多个用户共享，这一点与基表类似。视图是一个虚表，而基表是一个实表。虚表只有定义，没有对应的物理数据；而实表既有定义，又有对应的物理数据。虽然如此，视图一经定义就可和基表一样被查询、删除，还可用来定义新的视图。但更新(增、删、改)视图的操作有一定限制。视图给用户和应用程序提供了一种灵活的操纵数据的方式。

关于视图的详细讲解，请参见“第 5 章：创建数据库对象”。

Logs

日志是用于恢复目的的文件。日志记录了对数据库进行的每个 SQL 操作。万一发生故障，要将数据库恢复到某个一致的时间点，日志就显得至关重要了。数据库恢复日志保存对一个数据库所作的全部更改的记录，包括对新表的添加或对现存表的更新。这个日志由大量日志块组成，每一个日志块包含在名为日志文件的一个单独文件中。

数据库恢复日志可以用于确保故障(例如系统断电或应用程序出错)时不会使数据库处于不一致的状态。如果发生故障，就回滚已进行但未落实的更改，重新执行可能没有实际写入磁盘的所有已落实的事务。这些操作确保了数据库的完整性。关于使用数据库日志、备份和恢复的详细讲解，请参见“第 7 章：数据库备份与恢复”。

Bufferpool(s)

缓冲池是一块内存区域，所有索引和数据页(除了 LOB)都必须有序地经过该区域，从而进行处理。它是数据库管理器所使用的主要高速缓存。在数据库性能问题方面，缓冲池是进行调优的最重要的对象。关于缓冲池的规划设计请参见“第 3 章：创建数据库和表空间”。

系统编目表

每个数据库都会创建和维护一组描述数据的逻辑和物理结构的系统编目表。这些表包含有关数据库对象(表、视图、索引以及约束和数据库授权)的定义的信息，以及用户对这些对象所拥有的权限的安全性信息。这些表存储在 SYSCATSPACE 表空间中。它们在数据库创建时被创建，当创建、修改或者删除一个对象的时候，DB2 插入、更新或者删除描述对象的目录行。系统编目表是只读类型，因为它们是被 DB2 维护的。不能显式地创建或卸载它们，但是可以使用目录视图查询它们的内容。关于系统编目表的详细讲解，请参见“第 3 章：创建数据库和表空间”。

程序包

程序包是在准备包含编译以后的 SQL 语句和控制结构的程序的时候，所产生的一个对象，这些 SQL 语句和控制结构存在于单个源文件中，并且在运行中使用。程序包由段(section)组成。段包括编译以后的 SQL 语句。虽然每个段对应于单个语句，但是并非每个语句具有一个段。程序包和应用开发相关，本书中没有涉及到程序包的讲解。

性能视图

性能视图用于存储数据库管理员的性能和操作信息，应用程序可以使用这些信息。这些信息对于调试性能、诊断问题是非常有用的。性能视图可以用于提取数据库管理器特定对象和对象组的当前活动。性能视图存放在 DB2 的系统编目表空间中，关于这部分的详细信息，我们会在“第 9 章：DB2 性能监控”中详细讲解。

诊断文件

每个实例都有诊断文件，当数据库出现问题时，我们首先要检查诊断文件来判断数据库出现了什么类型的错误，关于数据库诊断的详细信息，我们会在“第 8 章：DB2 故障诊断”中详细讲解。

认证和权限

认证提供了一种用户验证机制，决定用户和密码能否连接实例或数据库。而权限决定该用户能否合法地存取数据。数据库相关的权限存储在数据库系统编目表中。关于这部分的详细信息，我们会在“第 13 章：数据库安全”中详细讲解。



创建实例和管理服务器

在安装完 DB2 数据库后，我们首先需要做的就是创建一个实例。因为要创建数据库就必须要先创建实例，数据库是运行在实例之上的。在本章中我们给大家讲解如何创建实例以及与实例相关的命令。DAS(Database Administrator Server)是一个特殊类型的实例，它主要执行远程管理任务。

本章主要讲解如下内容：

- 实例
- DAS(管理服务器)

2.1 实例

2.1.1 实例概念

从 DB2 体系结构的方面来看，实例实际上就是 DB2 的执行代码和数据库对象的中间逻辑层。实例可以看成是关于所有的数据库及其对象的逻辑集合，也可认为是所有的数据库及其对象和 DB2 的代码之间的联系和结合。实例为数据库运行提供一个环境。数据库在运行时，实例用来为数据库提供安全、通信、内存分配和进程间通信等功能。这样数据库只负责前台正常的运行，而一些后台的事情由实例来进行管理。实例对用户和开发人员来说是透明的。实例本质上由一组后台进程和共享内存组成。实例和数据库不一样的地方是，数据库是物理的，我们的表、索引存放在数据库中要占物理存储的；而实例是逻辑的，是共享内存、进程和一些配置文件(实例目录)的集合。当实例停止时，共享内存释放，进程停止。实例就相当于 Windows 中的服务的概念。所以大家在学习 DB2 时，首先要搞清楚实例的概念。

在实际生产系统中，我们可能需要创建多个实例来执行下列操作：

- 将一个实例用于开发环境，而将另一个实例用于生产环境
- 为一个特定环境调整实例
- 限制存取机密信息
- 控制为每个实例指定 SYSADM、SYSCTRL、SYSMAINT 和 SYSMON 权限
- 优化每个实例的数据库管理器配置
- 限制实例故障的影响。如果发生实例故障，则只有一个实例受影响，其他实例可以继续正常工作

当然，系统中的实例不是越多越好，如果在系统中创建的实例过多，不仅会造成额外的资源消耗(内存、硬盘空间等)，还会增大管理开销。过多的实例数量可能会有下面的影响：

- 每个实例都需要额外的系统资源(虚拟内存和磁盘空间)。
- 由于要管理其他的实例，因此增加了管理工作量
- 更多管理任务，因为要管理附加实例

注意：

在许多数据库产品中都有类似实例的这个概念，例如在 Oracle 中也叫实例(instance)，在 Informix 数据库中叫 Server 的概念，Sybase 和 SQL Server 中的 Server 概念也和实例类似。

2.1.2 创建实例

在 Windows 上 DB2 的安装过程中，如果没有其他实例的名称为“DB2”，那么将自动创建一个名为 DB2 的数据库管理器初始实例，该实例由 DB2INSTANCE 环境变量定义。如果安装了 DB2 V8，并且已升级到版本 9.1 或版本 9.5，那么默认实例为“DB2_01”。在 Windows 上创建一个实例的时候，不需要创建用户，创建完实例后，实例会作为一个服务存在。

而在 Linux 和 UNIX 上，要想创建一个实例，必须首先创建和实例名一样的用户及该用户所属的组。之所以需要创建用户，主要是因为需要该用户的 home 目录来作为实例目录，以存放实例相关的实例目录结构。

实例可以在 DB2 向导安装期间创建，但业务需求可能需要我们手工创建其他实例。创建实例使用 db2icrt(db2 Instance CReaTe)命令，db2icrt 命令的语法如图 2-1 所示。

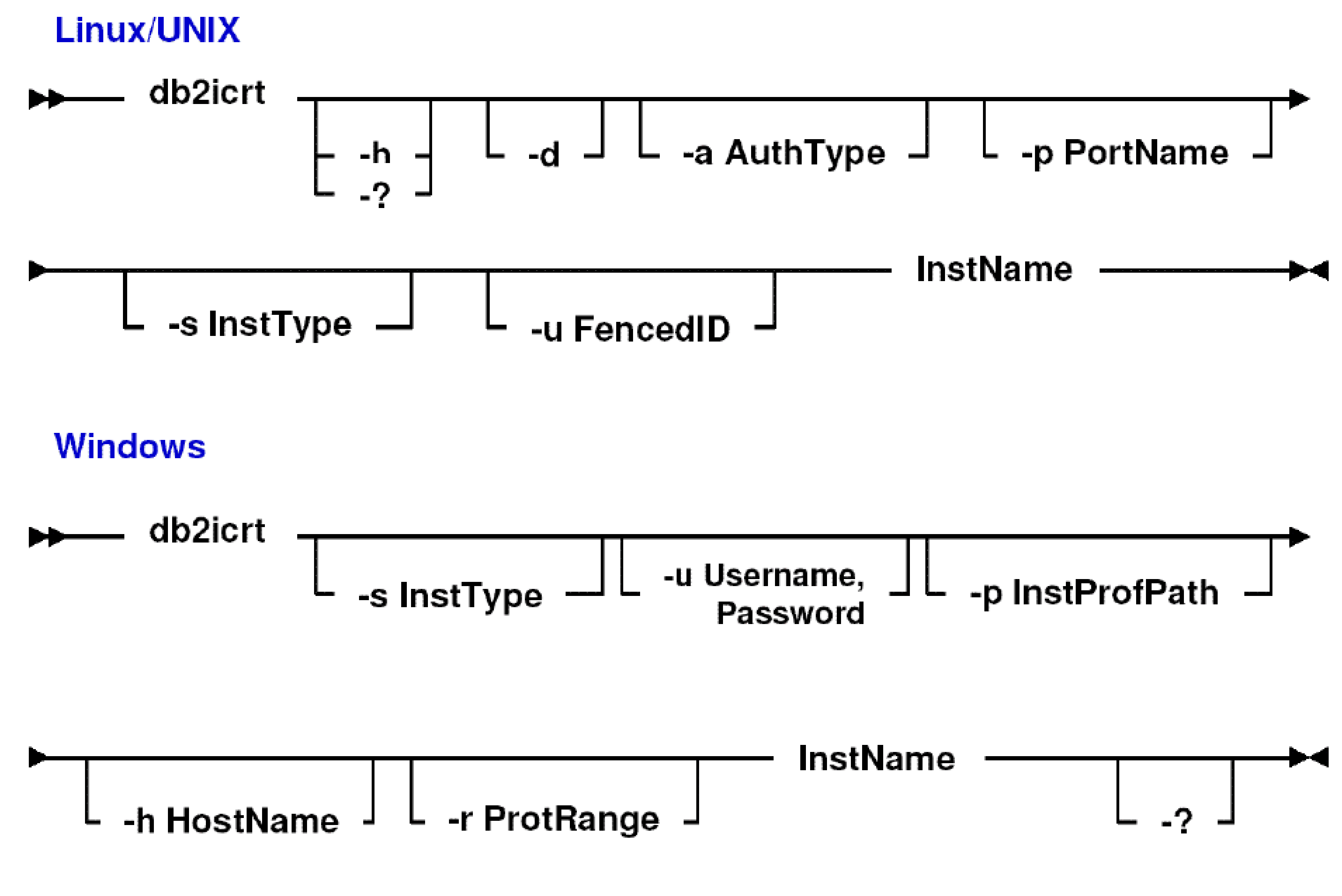


图 2-1　db2icrt 命令的语法

有一点我们需要特别注意，在 Linux 和 UNIX 上创建实例时，必须要有一个和实例同名的用户存在。如果该用户不存在，那么创建实例会报错而无法创建。如果用户存在，确保该用户未被锁定和密码未到期。

注意：

在 linux 和 UNIX 上创建实例时，必须创建和实例名一样的用户，而在 Windows 上不需要创建和实例同名的用户，但要确保创建的实例名与存在的服务名不相同，否则无法创建。

要使用 **db2icrt** 创建实例：

(1) 使用 root 权限登录(在 Windows 上使用系统管理员账号登录)。

(2) 首先利用操作系统命令(mkuser、useradd；mkgroup、groupadd)创建实例的用户和组，一般来说我们需要创建 2-1 所示的用户和组并设置密码。表 2-1 中的用户名和组名只是使用 DB2 安装向导期间 DB2 默认生成的。实际生产中，我们可以根据自己需要创建自己特定的组名和用户名，此处仅为说明作用。

表 2-1　默认生成的用户和组

用　　户	示例用户名	示 例 组 名
实例所有者	db2inst1	db2iadm1
受防护的用户	db2fenc1	db2fadm1

- 实例所有者 home 目录(假如为/home/db2inst1)是将在其中创建实例目录的位置。
- 受防护的用户(db2fenc1)用于在 DB2 数据库所使用的地址空间之外运行用户定义的函数(UDF)和存储过程。这个用户和应用开发有关，通常没有什么用，但是作为创建实例它是必需的。很多初学者往往被这个用户所迷惑，其实你可以不创建这个用户，你也可以使用实例用户作为受防护的用户。但是从应用程序安全和维护角度而言，建议创建这个用户。

(3) 运行 db2icrt 命令。例如，在 Linux 或 UNIX 操作系统上：

```
DB2DIR/instance/db2icrt -a AuthType -u FencedID InstName
```

在 Windows 操作系统上：

```
DB2DIR\bin\db2icrt InstName
```

其中 DB2DIR 是 DB2 安装目录。在 Linux/UNIX 操作系统上，默认 DB2 安装目录是/opt/IBM/db2/V9.5。

-a AuthType(Linux 或 UNIX)

表示实例的认证类型。AuthType 可为 SERVER、CLIENT、SERVER_ENCRYPT 和 DCS_ENCRYPT 其中之一。SERVER 是默认值。此参数是可选的，这些认证类型和安全有关，关于这部分的详细内容，我们在第 13 章安全部分会深入讲解。

-u FencedID

表示将用来运行受防护用户定义的函数(UDF)和受防护存储过程的用户的名称。这个用户和应用开发有关，虽然通常用不到，但是创建实例是必需的。

InstName

表示实例的名称。实例的名称必须与拥有实例的用户的名称相同。指定您创建的拥有实例的用户的名称。将在拥有实例的用户的主目录中创建该实例。

例如，如果正在使用服务器认证，受防护用户为 db2fenc1，并且拥有实例的用户为 db2inst1，那么使用以下命令以在 AIX 系统上创建实例：

```
/opt/IBM/db2/V9.5/instance/db2icrt -a server -u db2fenc1 db2inst1
```

db2icrt 命令除了上述必需选项外，还有一些可选选项，如下所示。

- –s 指定所创建的实例的类型，有以下几种类型：
  - ◊ ese 用于创建具有 DPF 支持的 DB2 数据库服务器的一个实例，该数据库服务

器带有本地和远程客户机。此类型是 DB2 企业服务器版的默认实例类型。

◊ wse　用于创建 DB2 数据库服务器的一个实例，该数据库服务器带有本地和远程客户机。此类型是 DB2 工作组版、DB2 易捷版或 Express-C 版的默认实例类型。

◊ standalone　用于创建 DB2 数据库服务器的一个实例，该数据库服务器带有本地客户机。此类型是 DB2 个人版的默认实例类型。

◊ client　用于创建 IBM 数据库服务器客户机的一个实例。

- –p　用于指定实例概要文件路径，对于环境变量 DB2INSTPROF。
- –u　用于指定 DB2 服务的账户名和密码。创建 ese 实例时需要此选项。
- –h　用于覆盖默认 TCP/IP 主机名。在创建默认节点(节点 0)时，将使用该 TCP/IP 主机名。
- –r　用于指定当在 MPP(数据库分区)方式下运行时，分区数据库实例要使用的一系列 TCP/IP 端口。如果指定了此选项，那么本地机器的 services 文件将使用下列条目进行更新：

```
DB2_InstName         baseport/tcp
DB2_InstName_1       baseport+1/tcp
DB2_InstName_2       baseport+2/tcp
DB2_InstName_END     endport/tcp
```

2.1.3　实例目录

一个实例创建后，会生成一个实例目录，实例目录存储着与一个数据库实例相关的所有信息。实例目录一旦创建，就不能更改其位置。在 Linux/UNIX 中为了拥有实例目录，必须创建和实例名相同的用户，其最终目的是为了用这个用户的 home 目录来作为实例目录。

实例目录包含：

- 数据库管理器配置文件(db2systm)
- 系统数据库目录(SQLDBDIR)
- 节点目录(SQLNODIR)
- 节点配置文件(db2nodes.cfg)，db2nodes.cfg 文件用来定义参与 DB2 实例的数据库分区服务器。如果想要将高速互连用于数据库分区。
- 诊断文件、数据库错误日志等。

在 Linux 和 UNIX 操作系统上，实例目录位于 INSTHOME/sqllib 目录中，其中 INSTHOME 是实例所有者的主目录，图 2-2 是实例 db2inst1 的实例目录。

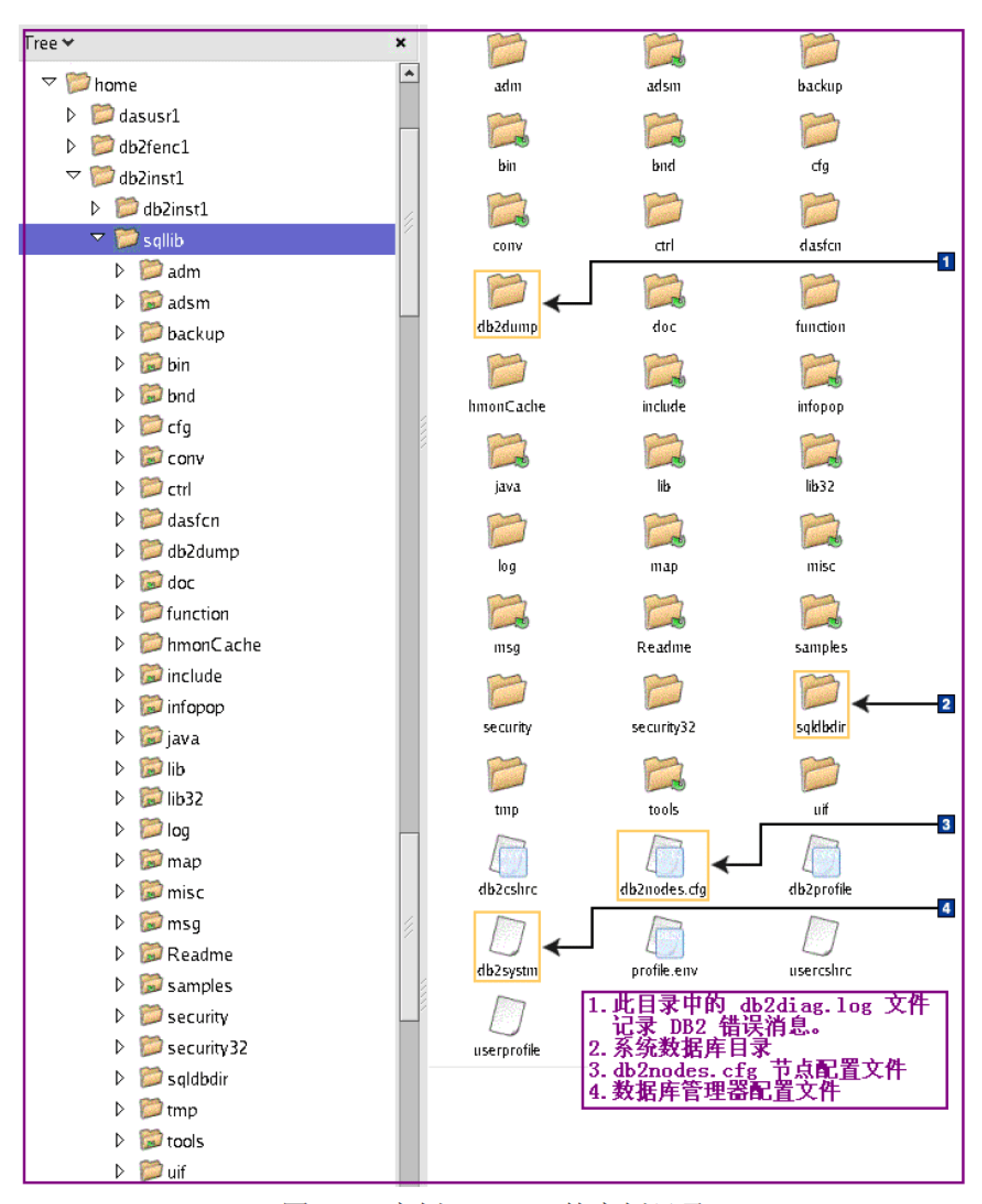


图 2-2　实例 db2inst1 的实例目录

在 Windows 操作系统上，实例目录位于安装了 DB2 数据库产品的目录下。实例名与服务名称相同，因此应该不会发生冲突。实例名不应与别的服务名称相同。您必须要有创建服务所需的正确权限。可在 DB2PATH 中使用 DB2INSTPROF 环境变量更改实例目录的位置，这需要该实例目录的写访问权。如果想要在不同于 DB2PATH 的路径中创建目录，那么输入 db2icrt 命令之前必须设置 DB2INSTPROF。

实例目录非常重要，下面我们举一个例子来说明实例目录。在讲这个例子之前我们先讲解一下 db2set 命令，之所以讲解这个命令是因为我们特殊定制实例时，需要用到这个命令。我们都知道在操作系统中我们可以使用 set、setenv 或 export 命令来修改操作系统环境变量。同样 DB2 实例本身也有实例级别的注册变量，为了修改这些默认的变量，我们使用

db2set 命令。它是 set 前加上 db2 表示设置 DB2 级别的变量。这个命令使用很简单。例如：

要查看已经设置的注册表变量，请从命令行执行下面这个命令：

```
db2set -all
```

您可能会得到类似下面这样的输出：

```
C:\>db2set -all
[e] DB2PATH=C:\Program Files\IBM\SQLLIB
[i] DB2INSTPROF=C:\DOCUMENTS AND SETTINGS\ALL USERS\APPLICATION DATA\IBM\
DB2COPY1
[g] DB2ADMINSERVER=DB2DAS00
```

正如您可能已经猜测到的那样，[i]表明该变量是在实例级上定义的，而[g]表明它是在全局级对该系统上所有实例上定义的；[e]表示是操作系统级别的环境变量。

要查看可以在 DB2 中进行定义的所有注册表变量，请使用下面这个命令：

```
db2set -lr
```

要在全局级上设置特定变量(在下面这个示例中为 DB2INSTPROF)的值，请使用：

```
db2set DB2INSTPROF="C:\INSTDIR" -g
```

要在实例级上为实例“DB2”设置变量，请使用：

```
db2set DB2INSTPROF="C:\MY FILES\SQLLIB" -i DB2
```

请注意上面的示例，在两个级别(实例级和全局级)上设置了同一个变量。当同一个注册表变量在不同级别上进行定义时，DB2 总是会选择最低级别的值；在本例中，它将选择实例级的值。

注意：

db2set 命令等号(=)前后不该留有空格。某些注册表变量为了使更改生效，要求您停止和启动实例(db2stop/db2start，这两个命令后面会讲解)。另一些注册表变量则没有这个需求。为了安全起见，建议您在对注册表变量作了更改后总是停止和启动实例。

好，现在我们已经知道 db2set 命令的用法，现在让我们先设置 DB2INSTPROF 注册变量，然后在 Windows 上创建一个新的实例并查看这个实例的实例目录。

```
C:\>db2set -all
[e] DB2PATH=C:\Program Files\IBM\SQLLIB
[i] DB2INSTPROF=C:\DOCUMENTS AND SETTINGS\ALL USERS\APPLICATIONDATA\IBM\
DB2COPY1
```

```
…………….略
C:\>db2set DB2INSTPROF=C:\INSTDIR  --重新设置 DB2INSTPROF 注册变量
C:\>db2icrt prod------------------创建 prod 实例
DB20000I  DB2ICRT 命令成功完成。
```

在我们创建完 prod 实例后，我们发现在 DB2INSTPROF 目录下生成了一个和实例同名的目录。这就是实例 PROD 的实例目录，如图 2-3 所示。

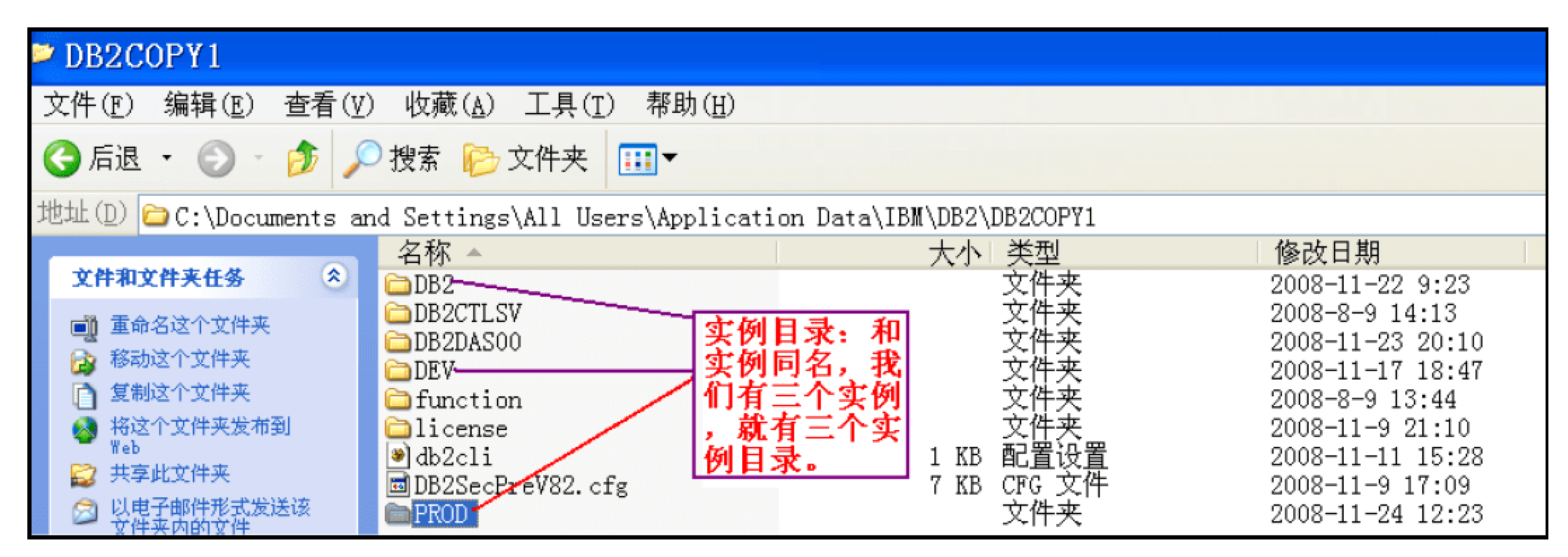


图 2-3 实例 PROD 的实例目录

我们进入我们刚刚创建的实例 PROD 的目录，如图 2-4 所示。

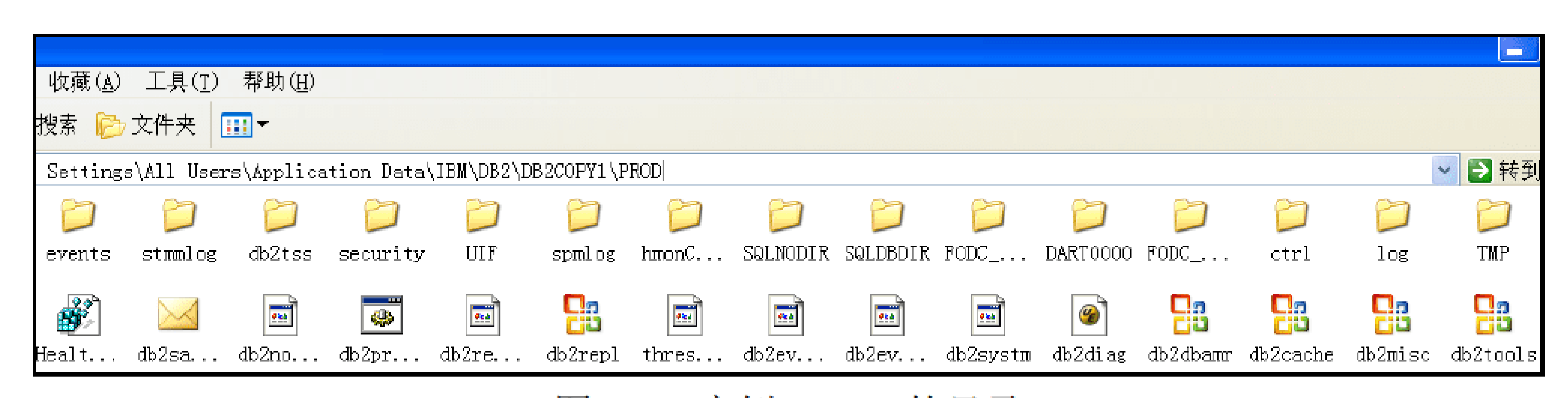


图 2-4 实例 PROD 的目录

在这个目录中存放实例的数据库管理器配置文件(db2systm)、系统数据库目录(SQLDBDIR)、节点目录(SQLNODIR)、节点配置文件(db2nodes.cfg)、诊断文件、数据库错误日志、安全配置等重要信息。所以实例目录至关重要。

同时 PROD 实例创建后，我们可以发现在 Windows 的服务面板中，多了一项 DB2COPY1-PROD 服务，如图 2-5 所示。

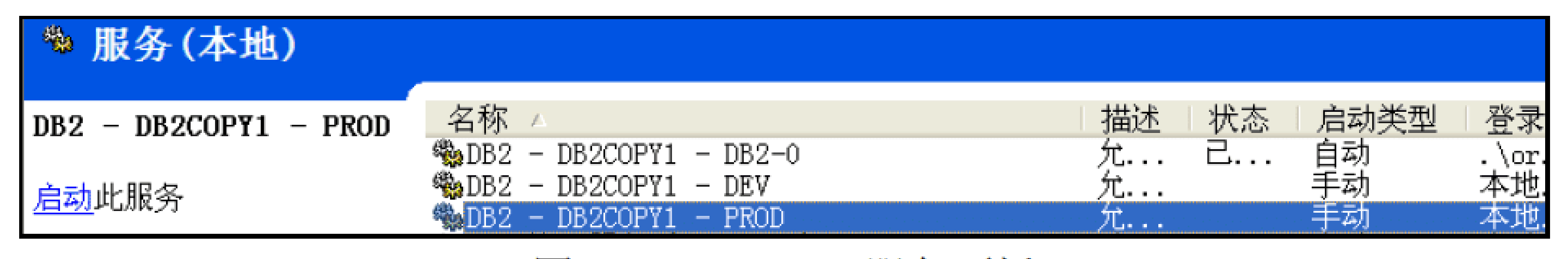


图 2-5 Windows 服务面板

这是因为在 Windows 上实例是作为服务存在的，而在 Linux/UNIX 上，实例是作为一组后台进程存在的。Linux/UNIX 上可以通过 db2_ps 或 ps –elf|grep –i INSTNAME 查看 DB2 进程的状态。

2.1.4　实例相关命令

在实例创建后，我们可以执行实例相关的命令来管理实例。在使用实例之前，必须更新每个用户的数据库环境，以便该环境可以访问实例并运行 DB2 实例相关命令。在运行这些命令之前我们首先要配置好实例的运行环境。这适用于所有用户(包括管理用户)。而且在执行这些命令时一定要确保具有足够的权限。实例相关的命令对权限要求很高，例如 db2icrt 和 db2idrop 需要 root 权限才能执行。而除了这两个命令，其他实例命令需要我们具有 SYSADM、SYSCTRL 或 SYSMAINT 的权限才能运行(第 13 章中有关于权限的详细讲解)。

1. 配置实例运行环境

我们都知道在 Linux 和 UNIX 环境中，在用户级上强制实施高安全策略时，与一个用户账户关联的文件和进程不能被其他用户直接访问。默认情况下，创建新的实例时，会在实例目录下生成一个特殊的 DB2 环境脚本 db2profile(Windows 下为 db2profile.bat)，每次实例所有者登录到系统时都要使用该文件配置其环境。这些脚本设置对数据库环境的访问，允许实例所有者执行 DB2 命令。为了让系统上的其他用户访问实例和 DB2 环境，他们也必须运行同样的脚本，否则将无法访问 DB2 实例运行环境。图 2-6 所示是一个由于没有设置 DB2 实例运行环境而导致无法执行相关 DB2 命令的例子。

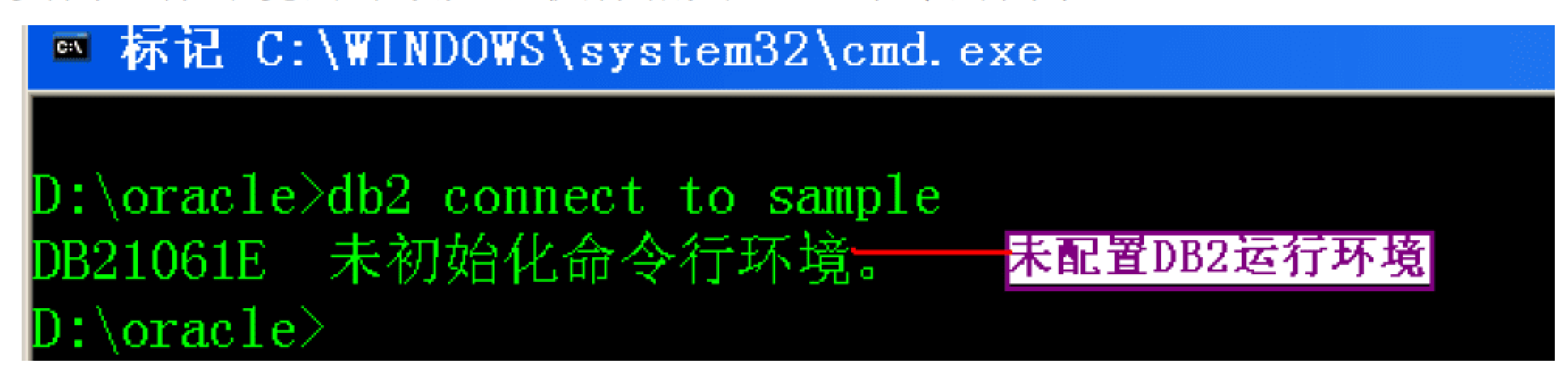


图 2-6　因没有设置 DB2 实例运行环境而无法执行相关 DB2 命令

我们可以通过设置 DB2 实例所有者的.profile 文件(或者该.profile 文件引用的 db2profile 文件)来配置其他用户的 DB2 运行环境。如果使用 Bourne 或 Korn shell，那么可以编辑目标用户账户的.profile 文件，使该用户登录到系统时自动运行 db2profile 脚本(创建实例时默认会加到实例用户的.profile 文件中)。对于 C shell 用户，可以编辑.login 文件让它运行 db2cshrc 脚本文件。为了选择要使用的实例的用户，请在该用户的.profile 或.login 脚本文件中添加下面的语句，或者在用户需要访问 DB2 的终端窗口中执行该语句(注意需要句点

(.)和空格):

```
Bourne 或 Korn shell: . INSTHOME/sqllib/db2profile ----注意前面的“.” INSTHOME
是实例目录
C shell: source INSTHOME/sqllib/db2cshrc
```

在配置好 DB2 的运行环境后，下面我们来讲解一些实例相关的命令。

列出实例

db2ilist 命令列出机器上的 DB2 实例。

```
C:\Program Files\IBM\SQLLIB\BIN>db2ilist
PROD
DEV
DB2
--系统存在三个实例。
```

迁移实例

```
db2imigr instanceName
        [/?]
        [/q]
        [/a:authType]
        [/p:instanceProf]
        [/u:username,password]
instanceName
        实例名
/?     此用法信息
/q     安静方式
/a     authType 是实例的认证类型(SERVER、CLIENT 或 SERVER ENCRYPT)
/p     instanceProf 是迁移实例的实例概要文件路径
/u     username,password 用于指定 DB2 服务的账户名和密码。在迁移分区实例时，此选项是
       必需的。
```

更新实例配置

如果通过安装“程序临时性修订(PTF)”或补丁更新了数据库管理器，那么应使用 db2iupdt 命令(需要 root 用户，在 DB2 V9.1 之后会在升级后自动调用此命令)来更新所有现有数据库实例。

db2iupdt 命令可在\sqllib\bin 目录中找到。要更新实例配置，请使用 db2iupdt 命令。按如下所示使用该命令：

```
db2iupdt InstName
      /u:username,password
```

```
          [/p:instance profile path]
          [/r:baseport,endport]
          [/h:hostname]
          [/?]
          [/q]
          [/a:authType]
    InstName 实例名
    /u       用于指定 DB2 服务的账户名和密码。当创建分区数据库实例时，此选项是必需的。
    /p       用于指定已更新的实例的新实例概要文件路径。
    /r       用于指定当在 MPP 方式下运行时，分区数据库实例要使用的一系列 TCP/IP 端口。如果
指定了此选项，那么本地机器的 services 文件将使用下列条目进行更新：
           DB2 InstName      baseport/tcp
           DB2_InstName_END   endport/tcp
    /h       用于覆盖默认 TCP/IP 主机名(如果当前机器有多个 TCP/IP 主机名)。
    /?       此用法信息
    /q       安静方式
    /a       authType 是实例的认证类型(SERVER、CLIENT 或 SERVER_ENCRYPT)
    示例：如果在创建实例后安装了 DB2 工作组服务器版或 DB2 企业服务器版，可输入以下命令来更
新该实例
       db2iupdt -u db2fenc1 db2inst1
```

注意：

db2imigr 和 db2iupdt 的区别是，db2iupdt 通常是小版本打补丁，而 db2imigr 通常是大版本迁移。例如从 DB2 V7 到 DB2 V8 用 db2imigr，而从 DB2 V8.1.4 到 V8.2.4 用 db2iupdt。

2. 自动启动实例

在 Windows 操作系统上，默认情况下，安装期间创建的数据库实例设置为自动启动。使用 db2icrt 创建的实例设置为手动启动。要更改启动类型，需要转至“服务”面板并在其中更改 DB2 服务的属性(手动或自动)。

在 UNIX 或 linux 操作系统上，要允许一个实例在每次系统重新启动后自动启动，请输入以下命令：

```
db2iauto -on <instance name>--其中<instance name>是实例的登录名
```

在 UNIX 或 Linux 操作系统上，要阻止一个实例在每次系统重新启动后自动启动，请输入以下命令：

```
db2iauto -off <instance name>--其中<instance name>是实例的登录名
```

3. 启动实例

在正常业务操作期间，可能需要启动或停止 DB2 数据库。例如，必须启动一个实例，然后才能执行下列某些任务：连接至该实例中的数据库、预编译应用程序、将程序包绑定至数据库或访问主机数据库。

在 Linux 或 UNIX 系统上启动实例之前需具有 SYSADM、SYSCTRL 或 SYSMAINT 权限的用户标识或名称进行登录；或者作为实例所有者登录。在 Windows 上启动实例，用户账户必须具有 Windows 操作系统定义的、用于启动 Windows 服务的正确特权。用户账户可以是 Administrators、Server Operators 或 Power Users 组的一个成员。启用了扩展安全性之后，默认情况下，只有 DB2ADMNS 和 Administrators 组的成员才能启动数据库。

使用命令行来启动实例，请输入：

```
db2start
```

在 Windows 上 db2start 命令将 DB2 数据库实例作为 Windows 服务来启动。通过在调用 db2start 时指定“/D”开关，仍可以在 Windows 上将 DB2 数据库实例作为进程运行。还可使用“控制面板”或 NET START 命令将 DB2 数据库实例作为服务启动。

4. 连接至实例和从实例断开

在所有平台上，要与另一个可能是远程的数据库管理器的实例连接，请使用 ATTACH 命令。要从实例断开，请使用 DETACH 命令。

要使用命令行来与实例连接，请输入：

```
db2 attach to <instance name>
```

例如，要连接至节点目录中先前编目的称为 testdb2 的实例：

```
db2 attach to testdb2
```

例如，在对 testdb2 实例执行维护活动后，要使用命令行从实例断开，请输入：

```
db2 detach
```

注意：

这两个命令我们在实际生产中很少用到，因为在 Linux/UNIX 环境中，我们每次都会用一个用户登录操作系统，这已经隐含地连接了实例。

5. 停止实例

我们有时可能需要停止数据库管理器的当前实例。要在 Linux 或 UNIX 系统上停止实

例，必须具有 SYSADM、SYSCTRL 或 SYSMAINT 权限的用户标识或名称登录或连接至实例；或者作为实例所有者登录。在 Windows 停止实例的用户账户必须具有 Windows 操作系统定义的正确特权。用户账户可以是 Administrators、Server Operators 或 Power Users 组的一个成员。在停止实例之前，要停止与数据库连接的所有应用程序和用户，要确保没有关键性的或极重要的应用程序在运行。

使用命令行来停止实例，请输入：

```
db2stop
```

如果停止期间仍然有应用连接，这时会报错，如图 2-7 所示。

```
E:\IBM\SQLLIB\BIN>db2stop
2008-11-24 15:52:48     0   0   SQL1025N  数据库管理器未停止，因为数据库仍在活
。
SQL1025N  数据库管理器未停止，因为数据库仍在活动。

E:\IBM\SQLLIB\BIN>
```

图 2-7　停止实例时仍有应用连接

如果我们强制所有应用程序和用户与该数据库断开。需要输入如下命令：

```
db2stop force
```

这时所有连接上数据库未提交的应用将强制回滚。

在 Windows 上除了使用命令行外，我们还可以通过“控制面板-服务”或 NET STOP 命令停止实例。

2.1.5　DB2INSTANCE 变量介绍

如果一个系统中有多个实例，那么如何在各个实例之间进行切换以及如何同时启动多个实例呢？这就需要使用 DB2INSTANCE 环境变量。环境变量是操作系统层面的，是在操作系统级别上定义的变量。最常使用的 DB2 环境变量是 DB2INSTANCE。该环境变量允许您指定当前活动实例，所有命令都将应用于该实例。例如，如果将 DB2INSTANCE 设置成“PROD”，那么发出命令“create database mydb”会创建出与实例“PROD”相关的数据库。但是如果您想创建与实例“DB2”相关的数据库，那么首先您必须将 DB2INSTANCE 变量的值更改成“DB2”。

可以使用控制面板(在 Windows 中)/db2profile(在 UNIX 中)来设置环境变量的值，将保证您下次打开窗口/会话时该值不变。但是，如果您想在给定的窗口/会话中临时更改该值，那么在 Windows 中您可以使用操作系统的“set”命令，在 UNIX 中可以使用“export”或“setenv”命令。例如，在 Windows 平台中，下面这个命令：

```
set DB2INSTANCE=DB2
```

会将环境变量 DB2INSTANCE 的值设置成“DB2”。**使用 set 命令时常犯的错误是在等号(=)前后留有空格。绝对不能有空格！**

要查看该变量的当前设置，您可以使用下面三个方法中的任何一个：

```
echo %DB2INSTANCE% (Windows only);echo $DB2INSTANCE(Linux/UNIX)
set DB2INSTANCE
db2 get instance
```

举例：假设一个系统中有多个实例，下面我们来举例说明如何通过设置 DB2INSTANCE 环境变量启动多个实例。

```
C:\Program Files\IBM\SQLLIB\BIN>db2ilist
PROD
DEV
DB2
--系统存在三个实例。
C:\Program Files\IBM\SQLLIB\BIN>set DB2INSTANCE
DB2INSTANCE=DB2
--当前活动实例是 DB2 实例。
C:\Program Files\IBM\SQLLIB\BIN>set DB2INSTANCE=PROD
C:\Program Files\IBM\SQLLIB\BIN>set DB2INSTANCE
DB2INSTANCE=PROD
C:\Program Files\IBM\SQLLIB\BIN>db2 get instance
 当前数据库管理器实例是：PROD
--db2 get instance 命令用来判断当前在哪个实例下。
C:\Program Files\IBM\SQLLIB\BIN>db2start
这时启动的是 prod 实例。
```

其实 DB2INSTANCE 环境变量类似 Oracle 中的变量 ORACLE_SID、Informix 中的变量 INFORMIXSERVER。主要用于在多个实例间进行切换。这个变量在 Windows 中常用，在 Linux/UNIX 中由于每个实例都有和它同名的用户，所以当我们用这个用户登录时已经隐含地连接了这个实例。所以相对来说在 Linux/UNIX 上很少用到这个变量。

2.1.6 删除实例

要删除实例，必须具有 root 权限，在 Windows 上必须具有系统管理员权限。删除实例之前确保所有的应用已经断开实例并且实例已经停止。

使用命令行除去实例，请输入：

```
db2idrop <instance_name>
```

db2idrop 命令从实例列表中除去实例条目，并除去实例所有者 home 目录下的 sqllib 子目录。所以删除实例时千万要小心，如有必要请在删除实例之前备份实例目录。

注意：

在 Linux 和 UNIX 操作系统上，试图使用 db2idrop 命令删除实例时，会生成一条消息，说明不能除去 sqllib 子目录，并且正在 adm 子目录中生成几个具有.nfs 扩展名的文件。adm 子目录是安装了 NFS 的系统，而这些文件在服务器上是受控的。必须从安装目录的文件服务器中删除*.nfs 文件，然后可除去 sqllib 子目录。

2.1.7　配置实例

每个实例创建后，都有一个实例配置文件(db2systm)，这个实例配置文件控制实例的安全、通信、管理和资源的分配。我们可以根据需要来查看、更改和复位这个配置参数。这个配置文件是二进制的，只能通过命令来修改。

可使用 db2 get dbm cfg 命令来查看当前实例配置参数。

要查看当前实例配置参数的当前值，请输入：

```
db2 get dbm cfg
```

它显示在安装该产品期间指定为默认配置参数的当前值，或在先前更新配置参数期间指定的那些值。

可在命令行使用 update dbm config 来更新实例配置文件。要更新实例配置文件中的个别条目，请输入：

```
db2 update dbm cfg using ..
```

要将配置参数复位为建议的默认值，请输入：

```
db2 reset dbm cfg
```

在某些情况下，对实例配置文件的更改仅在将更改装入内存后才生效(即在执行 db2stop 之后，再执行 db2start 时生效)。

关于实例配置文件，这超出了我们本章的讨论范围，我们会在后面的章节中再给大家详细讲解如何合理地配置实例配置参数以使其稳定、安全、高效运行。

2.2　管理服务器

2.2.1　管理服务器概念

DAS 是数据库管理服务器(Database Administration Server)的缩写。它是一个驻留在数

据库服务器机器上的特殊实例。DAS 是一个特殊的 DB2 管理控制点，它用于仅帮助其他 DB2 服务器执行远程管理任务(其他远程 DB2 服务器需要安装 Database Administration Client，DB2 V9 以后更名为 DB2 Client)。DB2 管理服务器(DAS)响应来自远程 DB2 服务器管理工具和配置助手(CA)的请求。如果要使用“客户机配置助手”来发现(Discover)远程数据库或随 DB2 产品一起提供的图形工具(例如，控制中心或任务中心)，那么 DAS 必须正在运行。

DAS 与实例是一对多的关系，即 DB2 数据库服务器中只能有一个 DAS，但这个 DAS 可以同时管理多个实例，如图 2-8 所示。

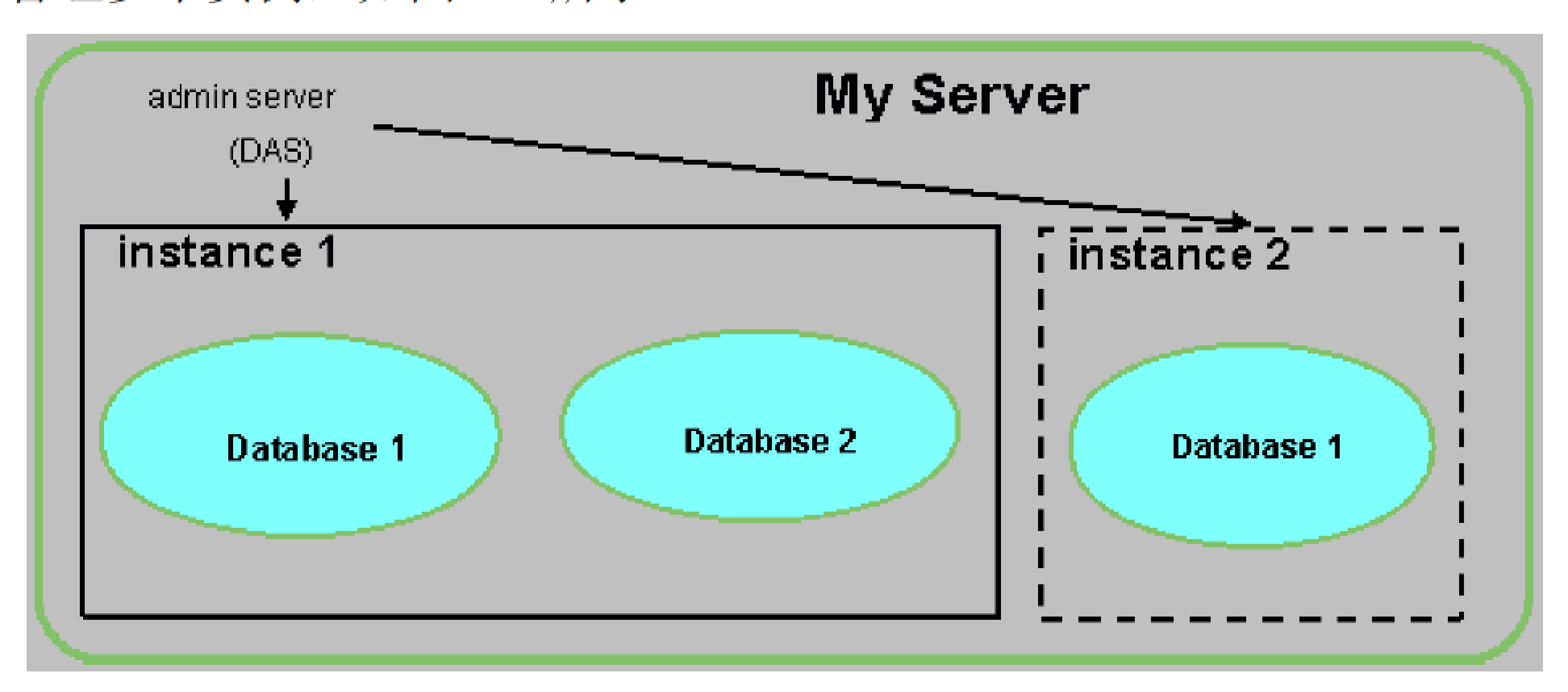


图 2-8　DAS 与实例的关系

DB2 全局注册变量参数 DB2ADMINSERVER 定义了管理 DB2 数据库服务器的 DAS：

```
E:\IBM\SQLLIB\BIN>db2set  -all |find /i "das"
[g] DB2ADMINSERVER=DB2DAS01
```

DAS 允许使用 DB2 工具(例如 Control Center、Client Configuration Assistant 和任务中心)对服务器上的数据库进行本地和远程管理。事实上，为了利用这些工具，DAS 实例必须存在并被启动。DB2 图形管理工具与 DAS 共同使诸如以下的任务得以被执行：

- 从远程客户机上管理、调度 DB2 或操作系统脚本以在服务器上运行
- 使用 DB2 Discovery 自动设置客户机/服务器连接
- 启动或停止远程 DB2 实例

举个例子，假设您安装了 DB2 的数据库服务器远在北京机房，而您现在想在上海利用 DB2 提供的图形化管理工具来管理这个北京机房的 DB2，那么您必须在北京的 DB2 服务器上配置启动管理服务器。其实 DAS 不是必须创建的，假设您不需要远程地使用图形化界面的工具来管理 DB2，那么您可以不创建 DAS，通过命令行远程地管理 DB2 不需要 DAS 存在。

2.2.2　创建管理服务器

管理服务器必须位于将要被管理和被搜索的每台服务器上。每个数据库服务器机器上只能创建一个 DAS 实例。除非您另外指定，否则这个实例会在 DB2 安装过程中自动建立。默认情况下，DAS 实例名在 Windows 上是 DB2DAS00，在 UNIX /Linux 上默认是 DB2AS。如果查看一下在 Windows 机器上运行的服务，您将看到列出了一个 DB2DAS00 服务，并且被配置成自动启动，如图 2-9 所示。

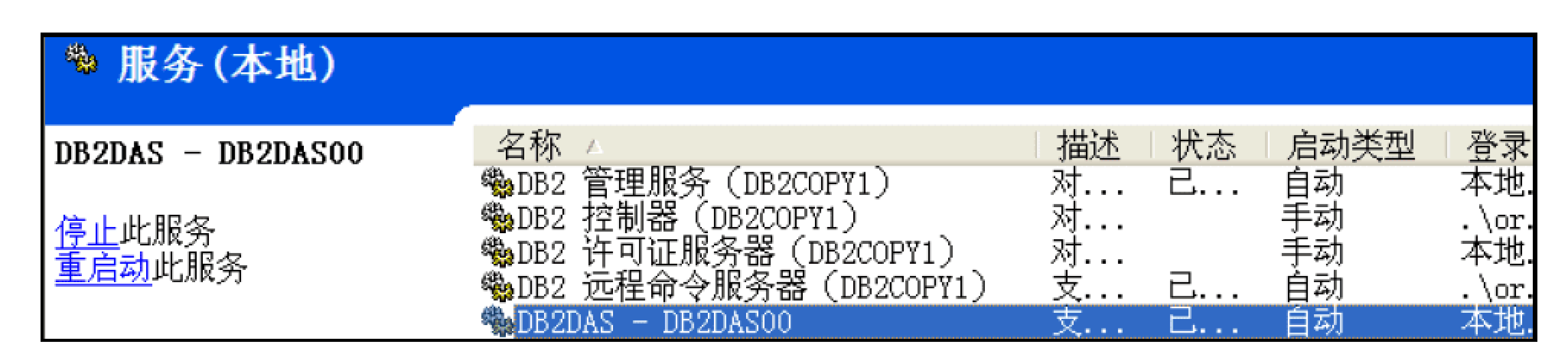


图 2-9　Windows 上默认创建的 DAS

如果在 DB2 安装过程中未自动创建 DAS 实例，那么我们必须手工创建它。在 UNIX/Linux 上，创建命令是 dasicrt name。在 Windows 上，命令是 db2admin create。

1. 在 Windows 上创建 DAS

在 Windows 上，创建 DAS 须具有本地 Administrator 权限。如果期望使用特定用户账户，在发出 db2admin create 时必须使用“/USER:”和“/PASSWORD:”。

在一个数据库服务器中只能有一个 DAS。如果已经创建了 DAS，那么需要通过发出 db2admin drop 来将其删除。

创建 DAS 后，可能需要更改 Windows 上运行 DAS 服务的用户标识。可使用 db2admin 命令设置或更改登录账户，如下所示：

```
db2admin setid <username> <password>
```

其中<username>和<password>是具有本地管理员权限的账户的用户名和密码。在运行此命令前，您必须使用具有本地管理员权限的账户或用户标识登录计算机。

注意：

在 Windows 上，不应使用控制面板中的服务实用程序来更改 DAS 的登录账户，因为这样将不会为登录账户设置某些必需的访问权。始终使用 db2admin 命令来设置或更改 DB2 管理服务器(DAS)的登录账户。

当安装过程与 DAS 相关时，作为在安装过程期间发生的情况的一个概述，考虑下列事项：

在 Windows 平台上：使用具有正确的服务创建权限的账户登录想要创建 DAS 的计算机。

当创建 DAS 时，可以选择是否提供用户账户名和用户密码。如果用户账户名和密码有效，它们将标识该 DAS 的所有者。不要使用为 DAS 创建的用户标识或账户名作为“用户账户”。将该账户名的密码设置为“密码永不到期”。在创建了 DAS 之后，就可以使用 db2admin setid 命令提供一个用户账户名和用户密码，来建立或修改它的所有权。

2. 在 Linux 和 UNIX 上创建 DAS

而在 Linux 和 UNIX 上，要想创建 DAS，必须首先创建和 DAS 一样的用户及该用户的组。之所以需要创建用户，主要是因为需要该用户的 home 目录来作为 DAS 实例目录存放 DAS 相关的实例目录结构。

要使用 dascrt 创建实例：

(1) 使用 root 权限登录(在 windows 上使用系统管理员账号登录)。

(2) 首先利用操作系统命令(mkuser、useradd；mkgroup、groupadd)创建 DAS 的用户和组并设置密码，一般来说我们需要创建表 2-2 所示的用户和组。表 2-2 中的用户和组只是使用 DB2 安装向导期间 DB2 默认生成的。实际生产中，我们可以根据自己需要创建自己特定的组名和用户名，此处仅为说明作用。

表 2-2 默认生成的用户和组

用　　户	示例用户名	示 例 组 名
DB2 管理服务器用户	dasusr1	dasadm1

(3) 运行 dascrt 命令。例如，在 Linux 或 UNIX 操作系统上：

```
DB2DIR/instance/dascrt -u DASuser
```

其中 DB2DIR 是 DB2 安装目录。在 Linux/UNIX 操作系统上，默认 DB2 安装目录是 /opt/IBM/db2/V9.5。

2.2.3 管理服务器相关命令

启动和停止 DB2 管理服务器

在 Windows 上手动启动或停止 DAS，您必须首先使用属于 Administrators、Server Operators 或 Power Users 组的账户或用户标识登录至计算机。要在 Linux/UNIX 上手动启动或停止 DAS，账户或用户标识必须成为 *dasadm_group* 的一部分。*dasadm_group* 在 DAS 配置参数

中指定。要启动或停止 DAS，使用 db2admin start 或 db2admin stop 命令。

列示 DB2 管理服务器

使用 db2admin 命令列示计算机上的 DAS 的名称。

更新 DB2 管理服务器(Linux 和 UNIX)

在 Linux 和 UNIX 操作系统上，可以在一台机器上多次安装 DB2 数据库产品。DB2 管理服务器则只能有一个，但是，所有 DB2 副本都将使用它。DAS 可以从任何一个 DB2 副本创建，并将在与该 DB2 副本相同的修订包级别运行。如果对该 DB2 副本打了补丁，应发出 dasupdt 命令来更新 DAS。

要更新 DAS：

(1) 使用超级用户权限(通常作为“root”用户)登录到计算机。

(2) 发出 dasupdt 命令：

```
DB2DIR/instance/dasupdt
```

其中 DB2DIR 表示安装 DB2 副本的位置。一旦完成此命令，DAS 将在 DB2 副本的新修订包级别运行。

2.2.4　删除 DB2 管理服务器

使用 db2admin 或 dasdrop 命令可以从 Windows 或 Linux/UNIX 平台删除 DAS。

要除去 DAS，在 Windows 系统确保有正确权限登录。在 Linux/UNIX 操作系统上首先作为具有 DASADM 权限的用户登录。使用 db2admin stop 来停止 DAS。

要删除 DAS，作为 root 用户登录，并使用 dasdrop 命令除去 DAS，如下所示：

```
dasdrop
```

注意：

dasdrop 命令除去 DB2 管理服务器的主目录下的 das 子目录。

2.2.5　配置管理服务器

可使用 DB2 get admin cfg 命令来查看 DAS 的当前配置参数。

要查看与 DAS 相关的 DB2 管理服务器配置参数的当前值，请输入：

```
db2 get admin cfg
```

它显示在安装该产品期间指定为默认配置参数的当前值，或在先前更新配置参数期间

指定的那些值。

可在命令行使用 update admin config 来更新 DAS 配置文件，必须从与 DAS 具有相同安装级别的实例使用命令行。要更新 DAS 配置文件中的个别条目，请输入：

```
DB2 update admin cfg using
```

要将配置参数复位为建议的默认值，请输入：

```
db2 reset admin cfg
```

在某些情况下，对 DAS 配置文件的更改仅在将更改装入内存后才生效(即在执行 db2admin stop 之后，再执行 db2admin start 时生效；或对于 Windows 平台，在停止并启动该服务时生效)。在其他情况下，配置参数是可联机配置的(即不必重新启动 DAS 就可以使更改生效)。

关于 DAS 这个概念，主要希望大家了解 DB2 数据库中有这个概念，它主要执行远程图形化界面管理功能，这个概念在 DB2 中相对来说不是特别重要。在很多 UNIX/Linux 环境中，客户都是通过命令行来进行远程管理的，这种情况下可以不创建 DAS。



创建数据库和表空间

在创建完实例后，我们下一步要做的工作就是创建数据库和存放数据库对象的容器——表空间。

本章我们主要讲解如下内容：

- 创建数据库
- 表空间设计
- 缓冲池

3.1 创建数据库

DB2 数据库概念

在 DB2 中，一个 DB2 实例可以同时管理多个 DB2 数据库，而一个 DB2 数据库只能由一个 DB2 实例管理，DB2 数据库与 DB2 实例是一种松散耦合的关系。在 UNIX 或者 Linux 下，创建数据库所生成文件所属的用户和组都是 DB2 实例的所有者，即创建实例的用户和组。实例和数据库的关系如图 3-1 所示。

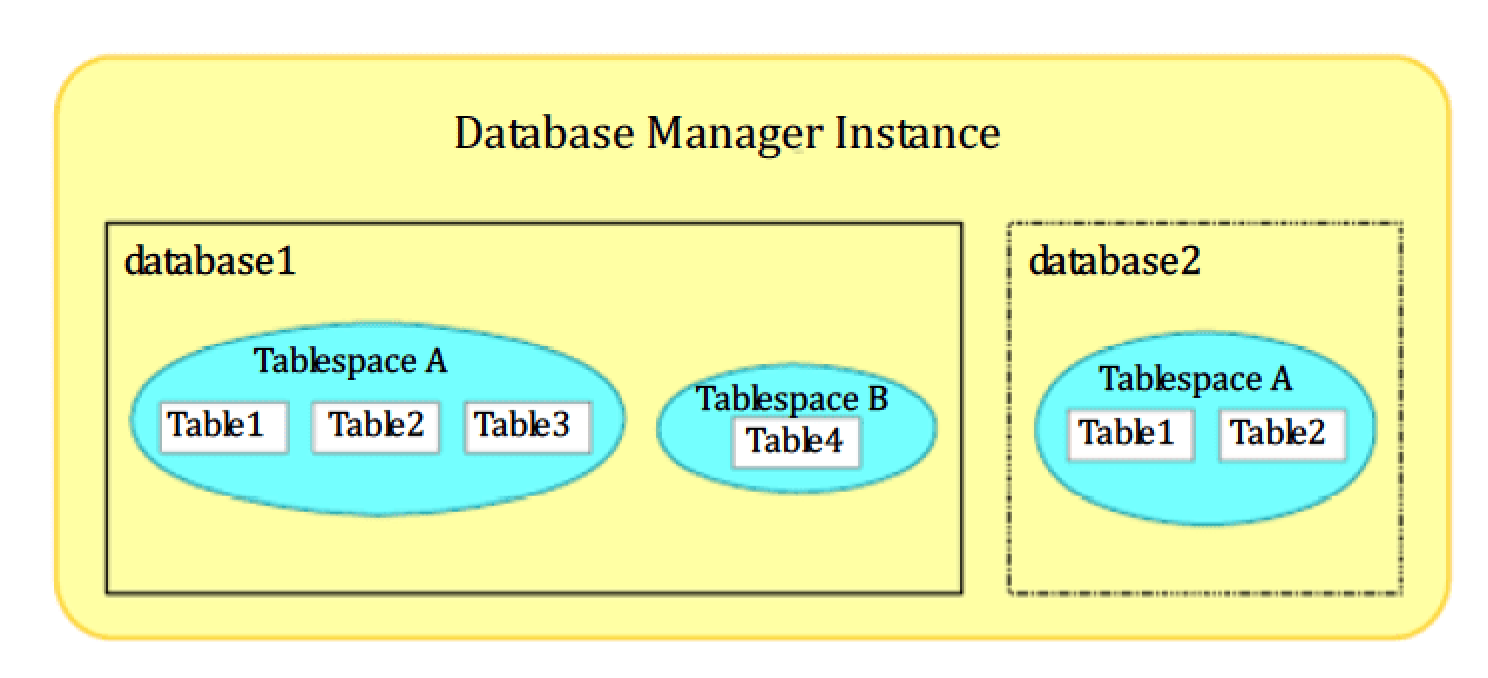


图 3-1　实例和数据库的关系

DB2 数据库实际上由一个对象集合组成。从用户的角度来看，数据库是一组通常以某种方式相关联的表。从数据库管理员 DBA 的角度来看，数据库比这要复杂一些。实际的数据库包含许多逻辑对象和物理对象：

- 表、视图、索引、模式、触发器、存储过程、程序包等数据库对象
- 缓冲池、日志文件、表空间
- 物理存储、表空间容器、目录、文件系统或裸设备

这些对象中的一部分(比如表或视图)帮助决定如何对数据进行组织；其他对象(比如表空间)涉及数据库的物理实现；最后，一些对象(比如缓冲池和其他内存对象)处理如何管理数据库性能；另外一些对象(比如日志文件)处理数据库的可恢复性。数据库的逻辑结构如图 3-2 所示。

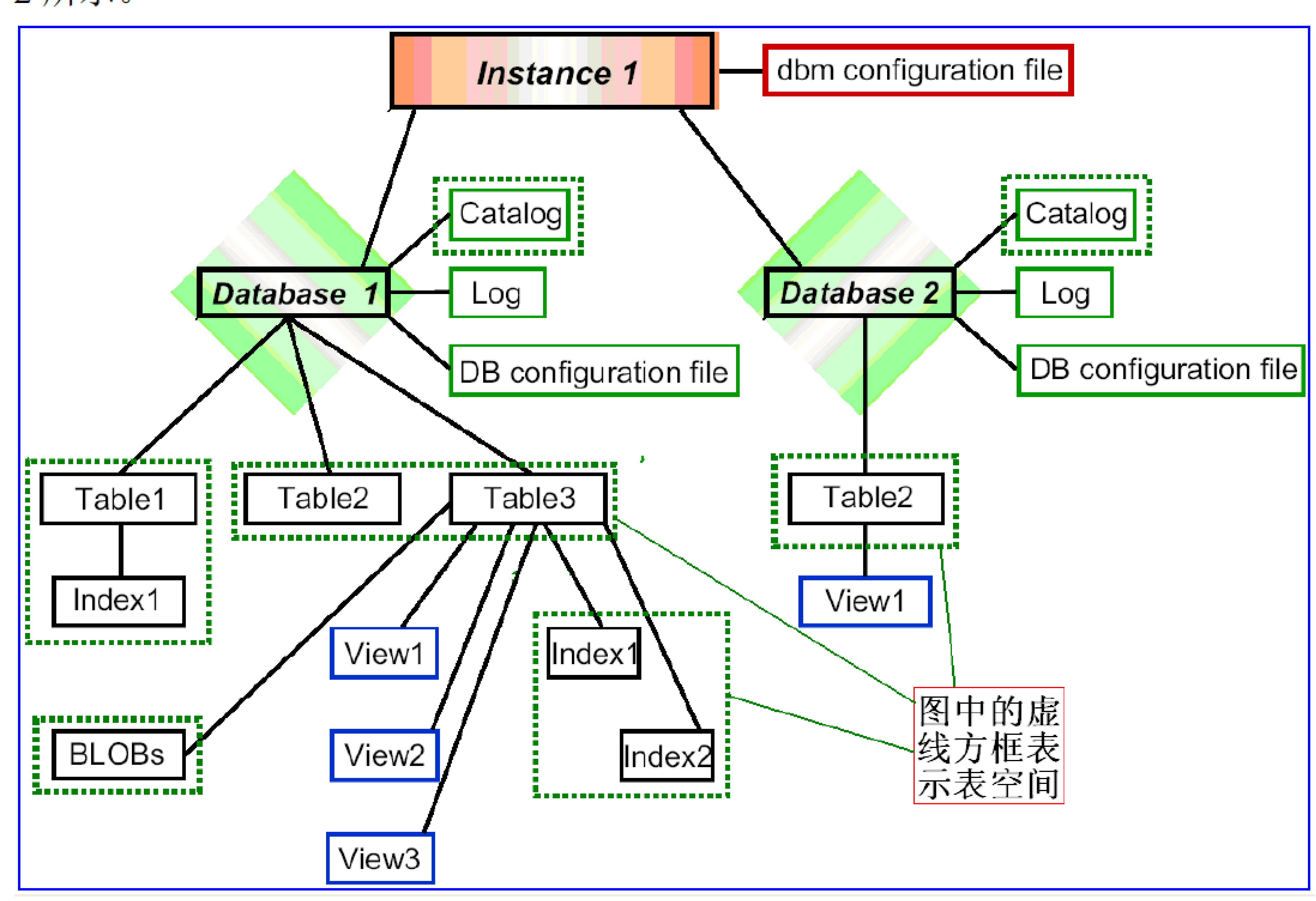


图 3-2　数据库的逻辑结构

DBA 应该首先关注数据库的物理设计，而不是直接研究所有可能的参数和对象组合。而数据库的逻辑设计，如表的字段属性设计，则主要由应用设计人员完成。DBA 核心工作之一就是研究如何创建数据库并分配它所需的磁盘存储。要正确地解决这个问题，需要了解数据库中的基本对象以及它们如何映射到物理磁盘存储。

3.1.1　DB2 数据库存储模型

DB2 利用一个逻辑存储模型和一个物理存储模型来处理数据。用户操作的实际数据放在表中。表由行和列组成，用户并不知道数据的物理表示。这一事实有时候称为数据的物理独立性。

表本身放在表空间中，表空间是存放表的储藏室(容器)，一个表空间可以包含多个表。同时，表空间物理上又对应着若干个表空间容器。容器可以由目录名、裸设备名或文件名标识。容器被分配给表空间。表空间可以跨许多容器，这意味着可以突破操作系统对于一个容器可以包含的数据量的限制。这样，表空间作为就逻辑设计中的表与物理设计中的容器之间的一个桥梁，表通过表空间实实在在地将数据存放到容器(文件或者目录)中。图 3-3 说明了所有这些对象之间的关系。

从图 3-3 中我们可以看到：一个数据库中有很多表空间，可以把数据库看做是很多个表空间的集合；我们可以根据需要创建多个表空间。而我们用户创建的表、索引等数据库对象存放在表空间中。表直接面向应用。而同时表空间又和底层的物理存储对应，表空间可以有多个容器，而容器是在底层存储上的。所以通过表空间数据库实现了物理存储和逻辑存储的统一。表空间是 DB2 中最重要的概念之一。

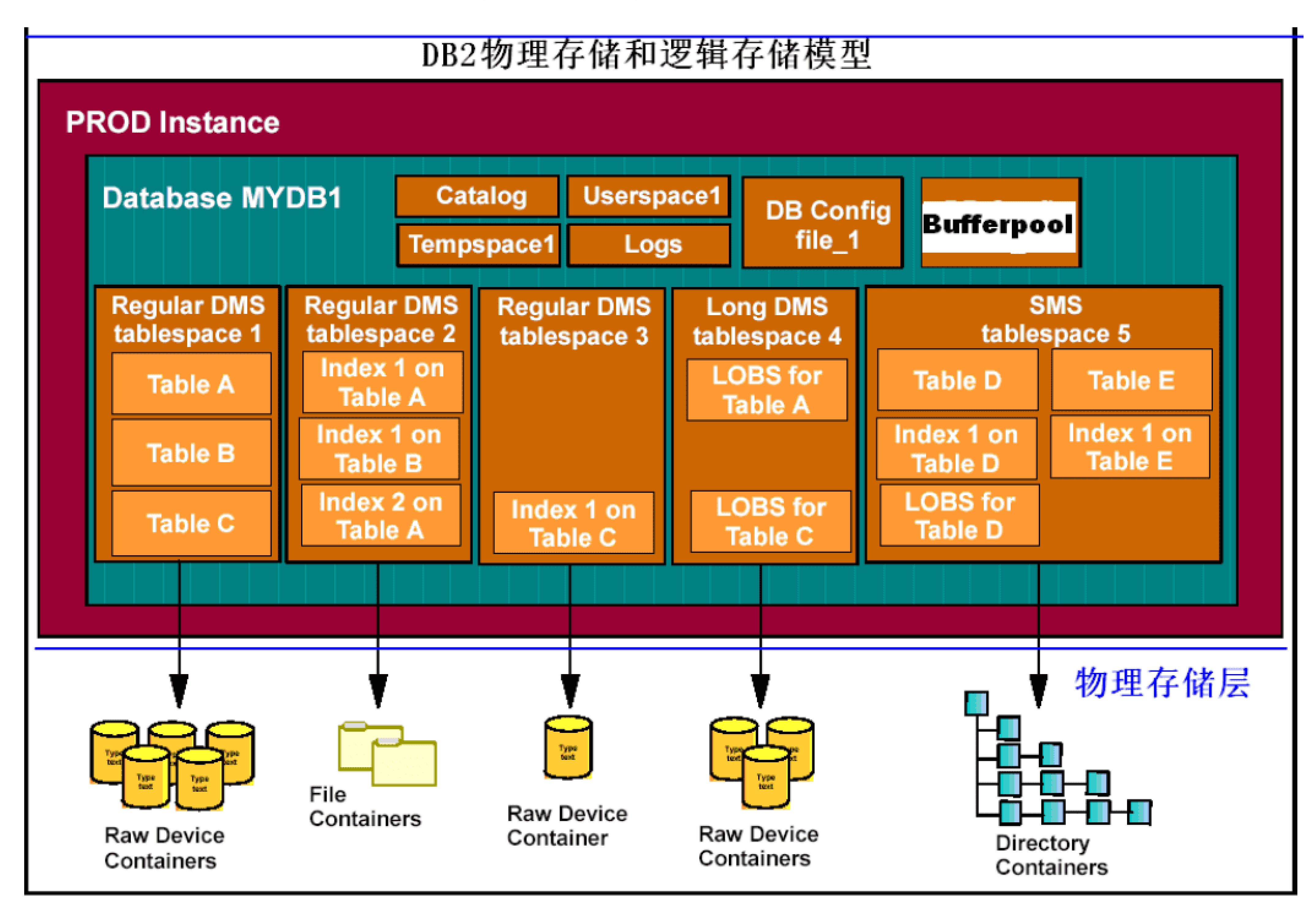


图 3-2　数据库、表空间、表和表空间容器之间的关系

下面我们先讲解数据库的物理存储。我们都知道操作系统的最小存储单位是块(block)，在 Linux 和 UNIX 上最小的块是 512 字节，在 Windows 上最小的存储单位为 1KB。而数据库中最小的存储单位是数据页(datapage)，它是 DB2 读写的最小单位。DB2 数据库中有 4KB、8KB、16KB 和 32KB 几种数据页。我们可以根据业务类型(OLAP、OLTP 等)和表的大小来选择合适的数据页。DB2 数据库在写物理存储时，为了保证写的吞吐量，引入了一个更大的单位 extent，它是整数倍的 datapage 的大小。这个我们可以在创建表空间时指定 extentsize 大小来确定。而表空间容器又是由很多个 extent 组成的。同时表空间又由很多容器组成。它们之间的关系如图 3-4 所示。

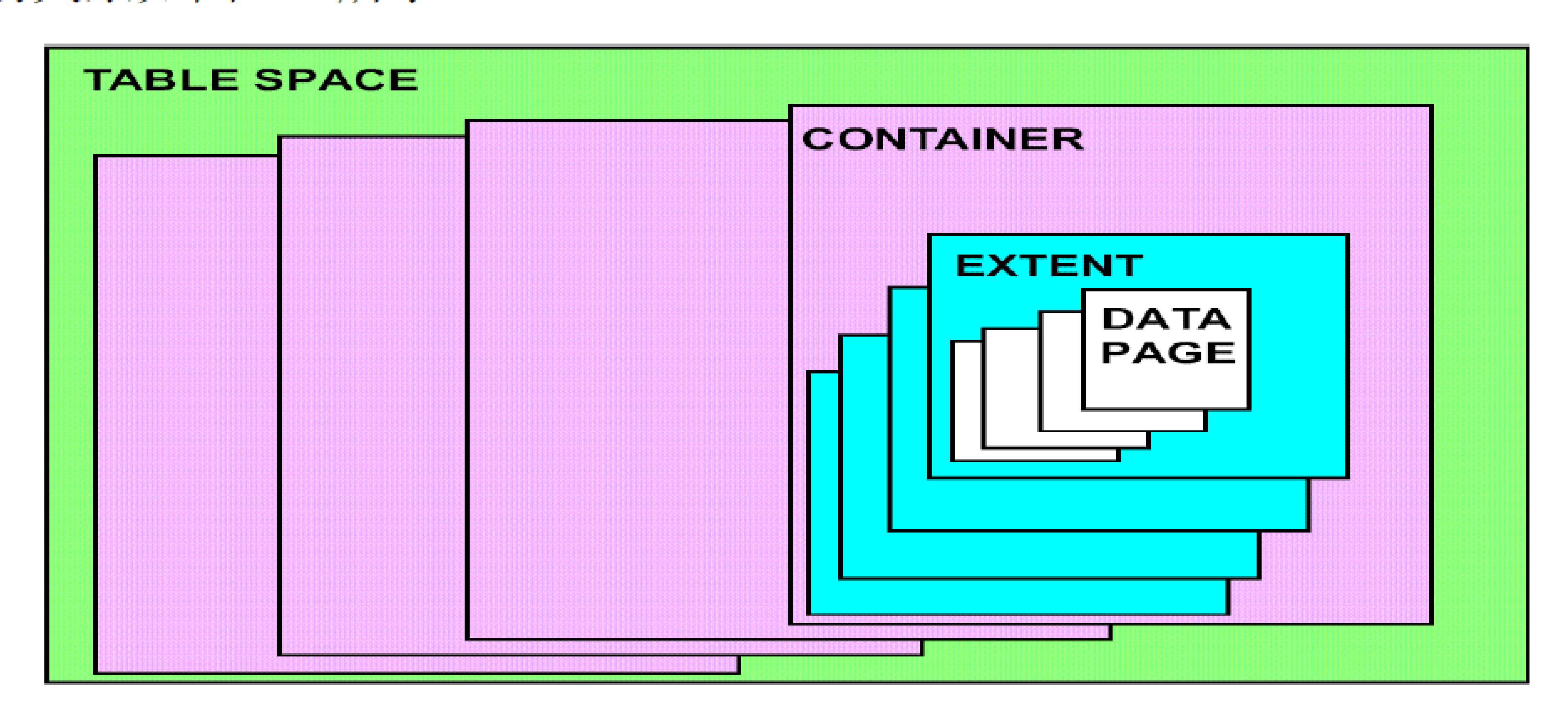


图 3-4　表空间、容器、extent 和数据页之间的关系

从上面的图 3-4 中我们知道我们创建的表最终是存储在底层表空间容器上的，那么 DB2 如何来写容器呢？请看下面的图 3-5。

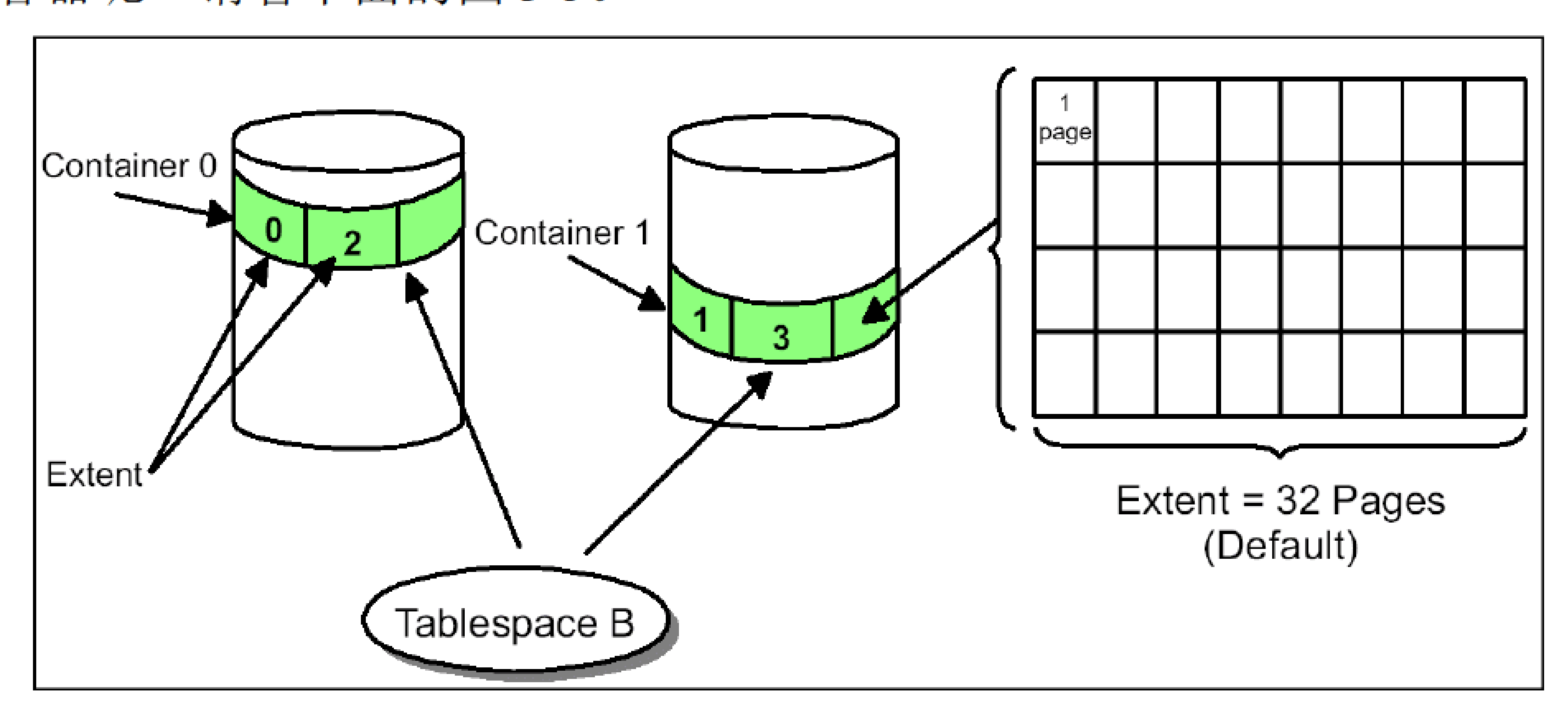


图 3-5　DB2 写容器的过程

我们知道表空间是由很多容器组成的，我们每次写容器的时候，写的单位为一个扩展数据块 extent。extent 的大小我们可以在创建数据库和表空间的时候通过 extentsize 大小指

定；而 extent 又是由很多 extentsize(默认 32)个数据页(datapage)组成。datapage 是 DB2 数据库中最小的存储单位，是我们每次读写的最小 I/O 单位。

图 3-6 显示具有两个 4KB 页扩展数据块(extent)大小的 HUMANRES 表空间，它有 4 个容器，每个容器有少量已分配的扩展数据块(extent)。DEPARTMENT 和 EMPLOYEE 表各有 7 页，且跨所有 4 个容器。一个 extent 同时只能被一个表写，也就是说，不可能两张表同时共用一个 extent，因为写容器的最小单位是 extent，当我们往表中插入数据发现没有空间时， DB2 就会为该表分配一个 extent。

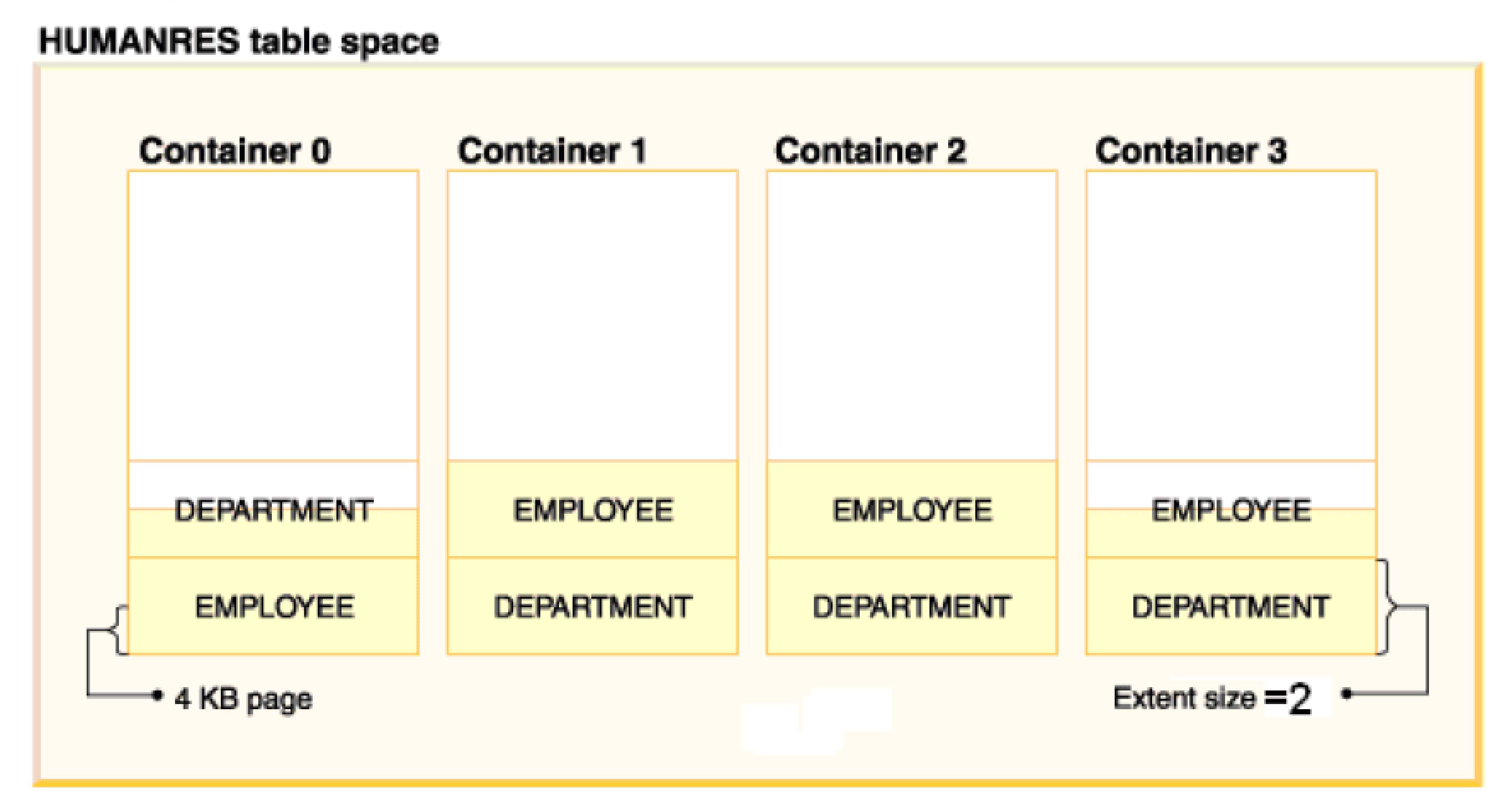


图 3-6 表空间中的容器、扩展数据块(extent)、数据页(datapage)和表空间之间的关系

通过上面的讲解，我们发现在数据库的物理存储和逻辑存储模型中，表空间连接了物理存储模型和逻辑存储模型，其扮演了一个承上启下的角色。在逻辑上，它向上面对的是数据库，向下它是存放表的容器，面向的是表；而同时表空间又在物理上映射底层的表空间容器——物理存储。同时表空间又是数据库性能调优的重点，而数据库创建工作的绝大部分都是围绕着表空间进行的，下面我们就首先讲解数据库中表空间这个最重要的概念。

我们首先了解表空间的类型，DB2 中有 3 种管理类型的表空间。

3.1.2　表空间管理类型

DB2 支持 3 种管理类型表空间：

- 系统管理的空间(System-Managed Space，SMS)：在这里，由操作系统的文件系统管理器分配和管理空间。在 DB2 V9 之前，如果不带任何参数创建数据库或表空间，就会导致所有表空间作为 SMS 对象创建。这种表空间依赖底层的操作系统(例如 AIX、HP-UX 或 Windows)来进行空间管理。

● 数据库管理的空间(Database-Managed Space，DMS)：在这里，由 DB2 数据库管理程序控制存储空间。表空间容器可使用文件系统或裸设备。
● DMS 的自动存储(Automatic Storage With DMS)：自动存储实际上不是一种单独的表空间类型，而是一种处理 DMS 存储的不同方式。因为数据库管理的空间 DMS 需要比较多的维护(见后面的 3.2.2 节)，在 DB2 V8.2.2 中引入了 DMS 自动存储，以简化表空间管理。

SMS 表空间需要的维护非常少。但是，与 DMS 表空间相比，SMS 表空间提供的优化选项少且性能不好。

那么，应该选择哪种表空间设计呢？下面我们来看看这几种表空间的比较。

1. DMS、SMS 与 DMS 自动存储比较

尽管下面的表 3-1 并不全面，但是它包含在 DMS、自动存储和 SMS 表空间之间进行选择时要考虑的一些因素。

表 3-1　DMS、自动存储和 SMS 表空间的比较

特　　性	SMS	DMS	自动存储
是否条带化(Striping)	是	是	是
默认类型	Version 8	无	Version 9
对象管理	操作系统	DB2	DB2
空间分配	按需增长/收缩	预先分配；大小可以收缩和增长，但是需要 DBA 干预	预先分配；可以自动增长
管理的简便性	最好；很少需要调优，甚至不需要	好，但是需要一些调优(例如 EXTENTSIZE PREFETCHSIZE)	最好；很少需要调优，甚至不需要
性能	不太好	很好；可通过利用裸设备多获得 5%到 10%的收益	最好；但是，不可以使用裸设备
表空间最大大小	64GB(4KB 页面)	2TB(4KB 页面)	2TB(4KB 页面)

SMS 表空间可以简化管理，DMS 可以提高性能，除此之外，这两种存储模型之间最显著的差异是表空间的最大大小。在使用 SMS 时，DBA 最多只能在表空间中放 64GB(4KB 页大小)的数据。将页面大小改为 32KB，可以将这个限制扩大到 512GB，但代价是每个页面上的可用空间可能会更少。DMS 模型会将表空间限制扩大到 2TB(4KB 页面大小的情况

下)。如果将页面大小改为 32KB，可用空间可以增长到 16TB。尽管还有让表大小突破 64GB 限制的其他方法，但是最简单的方法可能是一开始就使用 DMS 表空间。

2. DMS 与自动存储

DB2 V8.2.2 引入了自动存储的概念。自动存储允许 DBA 为数据库设置在创建所有表空间容器时可以使用的存储路径。DBA 不必显式地定义表空间的位置和大小，系统将自动地分配表空间。在 DB2 V9 中，数据库在创建时默认将启用自动存储，除非 DBA 显式地覆盖这个设置。

启用自动存储的数据库有一个或多个相关联的存储路径。表空间可以定义为“由自动存储进行管理”，它的容器由 DB2 根据这些存储路径进行分配。数据库只能在创建时启用自动存储。对于在最初没有启用自动存储的数据库，不能在以后启用这个特性。同样，对于在最初启用了自动存储的数据库，也不能在以后禁用这个特性。

下面的表 3-2 总结了管理 DMS 非自动存储和 DMS 自动存储之间的一些差异。

表 3-2　管理 DMS 非自动存储和 DMS 自动存储间的差异

特　　性	非自动存储	自 动 存 储
容器的创建	必须在创建表空间时显式地提供容器	不能在创建表空间时提供容器；它们将由 DB2 自动地分配
容器大小的调整	在默认情况下，表空间大小的自动调整是关闭的(AUTORESIZE NO)	在默认情况下，表空间大小的自动调整是打开的(AUTORESIZE YES)
初始大小	不能使用 INITIALSIZE 子句指定表空间的初始大小	使用 INITIALSIZE 子句指定表空间的初始大小
容器的修改	可以使用 ALTER TABLESPACE 语句(ADD、DROP、BEGIN NEW STRIPE SET 等等)执行容器修改操作	不能执行容器修改操作，因为由 DB2 控制空间管理
管理的简便性	可以使用重定向的恢复操作重新定义与表空间相关联的容器	不能使用重定向的恢复操作重新定义与表空间相关联的容器，因为由 DB2 控制空间管理

引入自动存储模型的主要目的是简化 DMS 表空间的管理，同时保持其性能特征。有的时候 DBA 必须定义使用的表空间的所有特征，但是许多应用程序都会从自动存储提供的简化管理获益。

3. DB2 存储模型小结

经过上面的讲解，我们对数据库、表空间、容器和数据库对象做个总结：

- 数据库是一个对象集合，包括表、索引、视图、大对象和触发器等；
- 这些对象存储在表空间中，表空间由表空间容器组成；
- 表空间可以由操作系统管理(SMS)，也可以由 DB2 管理(DMS、自动存储)；
- 表空间容器可以选择使用底层存储——文件系统、裸设备或操作系统目录；
- 表空间是由很多扩展数据块(extent)组成的，而 extent 又是由 extentsize(我们可自己定义)个数据页(datapage)组成的，数据页是最小的存储单位；
- 应该主要根据性能和维护因素决定要使用的表空间类型、扩展数据库大小、数据页大小和容器类型。

既然已经熟悉了不同类型的表空间，就该创建第一个数据库了。下面我们讲解如何创建数据库。

3.1.3 创建数据库

创建数据库有很多方法，我们可以选择在安装后打开“DB2 第一步”启动面板来创建数据库，这个我们已经在第 1 章讲过了。除此之外我们还可以通过 CREATE DATABASE 命令和创建数据库向导来创建数据库，下面我们分别讲解如何使用这两种方法创建数据库。

1. 使用命令创建数据库

从命令行创建 DB2 数据库是相当简单的。要创建数据库，必须调用 DB2 命令行处理程序(Command Line Processor，CLP)。调用方法是在 DB2 程序组的 Command Line Tools 文件夹中选择 Command Line Processor，或者从操作系统命令行执行命令 db2 cmd db2。

创建 DB2 数据库的语法如下：

```
CREATE DATABASE MYDB
```

您会问“就这么简单？”，是的，就这么简单！CREATE DATABASE 语句中唯一必需的选项就是数据库的名称。数据库的命名规则是：

- 数据库名称可以由以下字符组成：a-z、A-Z、0-9、@、#和$。
- 名称中的第一个字符必须是字母表字符、@、#或$；不能是数字或字母序列 SYS、DBM 或 IBM。注意，数据库名称不能超过 8 个字母。
- 数据库名称或数据库别名是一个唯一的字符串，包含前面描述的 1 个到 8 个字母、数字或键盘字符。

当然，有很多选项可供使用；不必只输入名称。我们来研究一下这个命令实际上会导致什么情况。

1) DB2 创建了什么

在发出 CREATE DATABASE 命令时，DB2 会创建许多文件。这些文件包括日志文件、配置信息、历史文件和 3 个默认的表空间。这些表空间是：

- SYSCATSPACE：这是保存 DB2 系统编目的地方，系统编目跟踪与 DB2 对象相关联的所有元数据，即通常所说的“数据字典”。
- TEMPSPACE1：DB2 用来放置分组、排序、连接和重组中间结果的临时工作区域。
- USERSPACE1：默认情况下存放所有用户对象(表、索引)的地方。

所有这些文件都放在默认安装路径的 DB2 目录中。默认安装路径通常是安装 DB2 产品的路径。

对于简单的应用程序，这个默认配置应该可以满足需要。但是，我们可能希望改变数据库文件的位置，或者改变 DB2 管理这些对象的方式。接下来，我们将更详细地研究 CREATE DATABASE 命令。

对于从 DB2V8 进行迁移的用户，有一个特殊的注意事项：在 DB2 V9 之前，CREATE DATABASE 命令会为上面列出的所有对象创建 SMS 表空间。在 DB2 V9 中，除了系统临时表空间外，所有表空间默认都将定义为自动存储(DMS)表空间。

2) CREATE DATABASE 命令

DB2 CREATE DATABASE 命令的完整语法很复杂，下面说明了 DBA 感兴趣的大多数选项。

```
>>-CREATE--+-DATABASE-+-database-name-+----------------------+-->
           '-DB-------'               '-| Database options |-'
```

数据库选项

```
CREATE DATABASE 选项:
 |--+----------------------------+------------------------------>
    '-AUTOMATIC STORAGE--NO|YES--'
>--+-------------------------------------------------+-------------->
   '-ON----+-path--+-+--+----------------------+-'
           '-drive-'    '-DBPATH ON--+-path--+-'
                                     '-drive-'
>--+----------------------+-------------------------------------->
   '-ALIAS--database-alias-'
>--+-------------------------------------------------+------------->
```

```
   '-USING CODESET--codeset--TERRITORY--territory-'
>--+-----------------------------------+------------------------>
   |              .-SYSTEM---------. |
   '-COLLATE USING--+-COMPATIBILITY--+-'
                  +-IDENTITY-------+
                  +-IDENTITY 16BIT-+
 >--+----------------------------------------+-------------------->
   '-CATALOG TABLESPACE--| tblspace-defn |-'
>--+------------------------------------+------------------------>
   '-USER TABLESPACE--| tblspace-defn |-'
>--+-----------------------------------------+------------------>
   '-TEMPORARY TABLESPACE--| tblspace-defn |-'
```

表空间选项

```
   tblspace-defn:
    |--MANAGED BY---------------------------------------------------->
   >--+-SYSTEM
USING--(----'container-string'-+--)-----------------------------+-->
      +-DATABASE
USING--(----+-FILE---+--'container-string'--number-of-pages-+--)-+
      |                '-DEVICE-'                                  |
      '-AUTOMATIC
STORAGE---------------------------------------------------------'
    >--+-----------------------------+------------------------------>
      '-EXTENTSIZE--number-of-pages-'
   >--+-------------------------------+------------------------------>
      '-PREFETCHSIZE--number-of-pages-'
   >--+---------------------+--+---------------------------------+->
      '-AUTORESIZE--+-NO--+-'  '-INITIALSIZE--integer--+-K|M|G-+-'
                    '-YES-'
   >--+------------------------------------+------------------------>
      '-INCREASESIZE--integer--+-PERCENT-+-'
                               '-+-K|M|G-'
   >--+-------------------------------+----------------------------|
      '-MAXSIZE--+-NONE-------------+-'
                 '-integer--+-K|M|G-'
```

下面我们将学习这些选项以及如何使用它们。

数据库位置

CREATE DATABASE 命令的参数之一是 ON path/drive 选项。这个选项告诉 DB2 希望在哪里创建数据库。如果没有指定路径，就会在数据库管理程序设置(DFTDBPATH 参数)

中指定的默认数据库路径上创建数据库。

```
C:\IBM\SQLLIB\BIN>db2 get dbm cfg |find /i "DFTDBPATH"
 默认数据库路径                              (DFTDBPATH) =C:
```

例如，以下的 CREATE DATABASE 命令将数据库放在 Windows 操作系统的 D 驱动器上的 DATA 目录中：

```
CREATE DATABASE MYDB ON D:\DATA
```

选择 Automatic storage(默认设置)允许 DBA 为数据库设置在创建所有表空间容器时可以使用的存储路径。DBA 不必显式地定义表空间的位置和大小，系统将自动地分配表空间。例如，下面的数据库创建语句将为数据库中的所有表空间设置自动存储。

```
CREATE DATABASE MYDB AUTOMATIC STORAGE ON
 /db2/mydbpath001,/db2/mydbpath002,/db2/mydbpath003
 AUTORESIZE YES INITIALSIZE 300M INCREASESIZE 75M MAXSIZE NONE
```

在 AUTOMATED STORAGE ON 选项后面，给出了 3 个文件目录(路径)。这 3 个路径是表空间的容器的位置。其他的选项是：

- AUTORESIZE YES：当表空间用光空间时，系统将自动地扩展容器的大小。
- INITIALSIZE 300M：没有定义初始大小的任何表空间的大小默认为 300MB。每个容器是 100 MB(因为有 3 个存储路径)。
- INCREASESIZE 75M(或百分数)：当表空间用光空间时，表空间的总空间增加 75MB。还可以指定一个百分数，在这种情况下，表空间会增长它的当前大小的百分数。假如为 20%，表示会比原来的空间增加 20%。
- MAXSIZE NONE：表空间的最大大小没有限制。如果 DBA 希望限制一个表空间可以占用的存储空间，那么可以指定一个最大值。

当使用 AUTOMATIC STORAGE 定义表空间时，不需要提供其他参数：

```
CREATE TABLESPACE TEST MANAGED BY AUTOMATIC STORAGE;
```

在这个命令中，可以提供与表空间相关联的任何参数；虽然使用自动存储可以大大简化日常的表空间维护，但是与重要的大型生产表相关联的表空间可能需要 DBA 更多地干预。

在没有启用自动存储的数据库中创建表空间时，必须指定 MANAGED BY SYSTEM 或 MANAGED BY DATABASE 子句。使用这些子句会分别创建 SMS 表空间和 DMS 表空间。在这两种情况下，都必须提供容器的显式列表。

如果数据库启用了自动存储，那么在定义表空间时还有另一个选择。可以指定 MANAGED BY AUTOMATIC STORAGE 子句，或者完全去掉 MANAGED BY 子句(这意味着自动存储)。在这种情况下，不提供容器定义，因为 DB2 会自动地分配容器。

代码页和整理次序

所有 DB2 字符数据类型(CHAR、VARCHAR、CLOB、DBCLOB)都有一个相关联的字符代码页。可以认为代码页是一个对照表，用来将字母数字数据转换为数据库中存储的二进制数据。一个 DB2 数据库只能使用一个代码页。代码页是在 CREATE DATABASE 命令中使用 CODESET 和 TERRITORY 选项设置的。代码页可以使用单一字节表示一个字母数字字符(单一字节可以表示 256 个独特元素)，也可以使用多个字节。英语等语言包含的独特字符相当少，因此单字节代码页(SBCS)对于存储数据足够了。东亚国家语言(中文、日文、韩文等)需要超过 256 个元素才能表示所有的独特字符，因此需要多字节代码页(通常是双字节代码页 DBCS)。

在默认情况下，数据库的整理次序根据 CREATE DATABASE 命令中使用的代码集进行定义。如果指定选项 COLLATE USING SYSTEM，就根据为数据库指定的 TERRITORY 对数据值进行比较。如果使用选项 COLLATE USING IDENTITY，那么以逐字节的方式使用二进制表示来比较所有值。

例如中文代码页为 1386，codeset 为 GBK，TERRITORY 为 CN。创建数据库时要注意选择合适的代码页，这些参数在数据库创建好后都不能再进行修改，务必要慎重选择。如果客户端访问数据库服务器代码页不一样，将无法访问。

对于需要使用 XML 数据的应用程序，有一个特殊的注意事项。当前，DB2 只在定义为 Unicode 数据库才能同时存储 XML 文档和 SQL 数据的更多传统格式，比如整数、日期/时间、变长字符串等等。随后，您将在这个数据库中创建对象来管理 XML 和其他类型的数据。如果数据库在创建时没有启用 Unicode 支持，就不能在其中创建 XML 列。

假如要创建一个同时支持 XML 和 SQL 的数据库，请执行如下命令：

```
create database xmldb using codeset UTF-8 territory us
```

一旦创建了 Unicode 数据库，您就不需要发出任何专门的命令或采取任何进一步措施来使 DB2 能够以它自身分层的格式存储 XML 数据和关系数据。

表空间定义

3 个表空间(SYSCATSPACE、TEMPSPACE1、USERSPACE1)都是在默认目录中自动创建的(ON 关键字)，除非指定它们的位置。对于每个表空间，DBA 可以指定表空间应该使

用的文件系统的特征。

3 个表空间使用以下语法进行定义：

```
>--+------------------------------------+------------------->
   '-CATALOG TABLESPACE--| tblspace-defn |-'
>--+--------------------------------+---------------------->
   '-USER TABLESPACE--| tblspace-defn |-'
>--+-----------------------------------------+----------------->
   '-TEMPORARY TABLESPACE--| tblspace-defn |-'
```

如果省略任何关键字，DB2 将使用默认值来生成表空间。表空间定义采用这些选项，其语法如下：

```
|--MANAGED BY--------------------------------------------------->
                    |>--+-SYSTEM
USING--(----'container-string'-+--)----------------------------------+-->
   '-DATABASE
USING--(----+-FILE---+--'container-string'--number-of-pages-+--)-'
'-DEVICE-'
   >--+-----------------------------+------------------------------->
   '-EXTENTSIZE--number-of-pages-'
   >--+------------------------------+------------------------------>
   '-PREFETCHSIZE--number-of-pages-'
```

注意，上面的语法不包括与自动存储数据库相关联的选项。

我们来详细看看这个语法。MANAGED BY 选项让 DB2 生成这些表空间并决定如何管理空间。SMS 表空间使用 SYSTEM USING 关键字，如下所示：

```
SYSTEM USING ('container string')
```

对于 SMS 表空间，容器字符串(container string)标识一个或多个将属于这个表空间的容器，表空间数据将存储在这些容器中。每个容器字符串可以是绝对的或相对的目录名。如果目录名不是绝对的，它就相对于数据库目录。如果目录的任何部分不存在，数据库管理程序就会创建这个目录。容器字符串的格式取决于操作系统。

使用 DATABASE USING 关键字定义 DMS 表空间：

```
DATABASE USING ( FILE/DEVICE 'container string' number of pages|K|M|G )
```

对于 DMS 表空间，容器字符串标识一个或多个将属于这个表空间的容器，表空间数据将存储在这些容器中。指定容器的类型(FILE 或 DEVICE)和大小(按照 PAGESIZE 大小的页面)。大小还可以指定为一个整数，后面跟着 K(表示千字节)、M(表示兆字节)或 G(表示千

兆字节)。可以混合指定 FILE 和 DEVICE 容器。

对于 FILE 容器，容器字符串必须是绝对或相对的文件名。如果文件名不是绝对的，它就相对于数据库目录。如果目录名的任何部分不存在，数据库管理程序就会创建这个目录。如果文件不存在，数据库管理程序就会创建这个文件并初始化为指定的大小。对于 DEVICE 容器，容器字符串必须是设备名而且这个设备必须已经存在，对于 DEVICE 容器通常需要使用操作系统 root 权限创建逻辑卷并且赋予 DB2 实例使用的权限，一般通过 UNIX/Linux 的 chown 命令实现这一点。

重要提示：所有容器必须在所有数据库上是唯一的；一个容器只能属于一个表空间。

```
EXTENTSIZE  number of pages
```

EXTENTSIZE 指定数据库可以写到一个容器中的 PAGESIZE 页面数量，达到这个数量之后将跳到下一个容器。EXTENTSIZE 值还可以指定为一个整数，后面跟着 K、M 或 G。数据库管理程序在存储数据时重复地循环使用各个容器。EXTENTSIZE 大小是在表空间级定义的。一旦为表空间指定了扩展数据块大小，就不能改变了。数据库配置参数 DFT_EXTENT_SZ 指定数据库中所有表空间的默认扩展数据块大小。这个值的范围是 2 到 256 个页面；因此，绝对大小是从 8KB 到 1024KB(对于 4KB 页面)，或者从 16 KB 到 2048KB(对于 8KB 页面)。可以在 CREATE TABLESPACE 语句中使用 EXTENTSIZE 参数覆盖这个数字。

```
PREFETCHSIZE  number of pages
```

PREFETCHSIZE 指定在执行数据预获取时将从表空间中读取的 PAGESIZE 页面数量。连续的预读取是指数据库管理程序能够提前预测查询，在实际引用页面之前读取这些页面。这样查询就不需要等待底层操作系统执行 I/O 操作。这种异步的检索可以显著减少执行时间。可以通过修改 CREATE TABLESPACE 语句中的 PREFETCHSIZE 参数来控制执行预获取的积极程度。在默认情况下，这个值设置为 DFT_PREFETCH_SZ 数据库配置参数。这个值代表在 DB2 触发预读取请求时每次读取多少个页面。通过将这个值设置为扩展数据块大小的倍数，可以并行地读取多个扩展数据块。当表空间的容器在不同的硬盘上时，这个功能甚至效率更高。预读取大小还可以指定为一个整数，后面跟着 K、M 或 G。

关于 PREFETCHSIZE 的设置，我们在后面表空间性能小节还会详细讲解。

CREATE DATABASE 命令示例

下面是一个 CREATE DATABASE 命令的示例，它使用了前面讨论的许多选项。

```
CREATE DATABASE MYDB
DFT_EXTENT_SZ 4
```

```
CATALOG TABLESPACE MANAGED BY DATABASE USING
  (FILE 'C:\DB2DATA\CATALOG.DAT' 2000, FILE 'D:\DB2DATA\CATALOG.DAT' 2000)
    EXTENTSIZE 8
   PREFETCHSIZE 16
TEMPORARY TABLESPACE MANAGED BY SYSTEM USING
  ('C:\TEMPTS','D:\TEMPTS')
USER TABLESPACE MANAGED BY DATABASE USING
   (FILE 'C:\TS\USERTS.DAT' 1200)
   EXTENTSIZE 24
   PREFETCHSIZE 48
```

我们来详细地看看每一行：

- CREATE DATABASE：这个语句定义要创建的数据库的名称。
- DFT_EXTENT_SZ 4：这个参数告诉 DB2 默认的扩展数据块大小是 4 个页面，除非在创建表空间时显式地声明，否则默认使用这个值。
- CATALOG TABLEPSACE MANAGED BY DATABASE USING：DB2 编目空间将由数据库管理。
- FILE 'C:\....'：表空间的位置将跨两个文件，每个文件有 2000 个页面的空间。
- EXTENTSIZE 8：EXTENTSIZE 是 8 个页面。这个设置会覆盖 DFT_EXTENT_SZ。
- PREFETCHSIZE 16：在查询处理期间，同时预读取 16 个页面。
- TEMPORARY TABLESPACE MANAGED BY SYSTEM USING：DB2 使用的临时空间将由操作系统处理。
- 'C:\TEMPTS' ...：临时空间将跨两个文件，文件的大小在 DB2 执行期间自动地调整。
- USER TABLESPACE MANAGED BY DATABASE USING：用户表空间(放置真正的表的地方)将由 DB2 直接管理。
- FILE 'C:\TS\...'：这个空间只有一个容器，它由 1200 个页面组成。
- EXTENTSIZE 24：USER 表空间的 EXTENTSIZE 是 24 个页面。
- PREFETCHSIZE 48：查询处理期间，同时预读取 48 个页面。

上面我们介绍了关于如何创建 DB2 数据库的背景知识。在大多数情况下，CREATE DATABASE 命令的默认值提供了一个可以满足开发和测试需要的数据库。一旦决定将数据库转入生产环境，就需要对 DB2 使用的数据布局和表空间定义付出更大的努力。尽管这需要做更多的规划工作，但是产生的数据库更容易管理，性能也可能更好。关于这部分内容我们会在 3.2 节中详细讲解。

2. 使用创建数据库向导创建数据库

如果感觉到通过上面命令创建数据库比较麻烦，您可以通过创建数据库向导来创建数据库，创建数据库向导将带领我们执行许多步骤来生成数据库。向导首先询问数据库的名称、创建它的默认驱动器(如果没有指定其他驱动器，就会使用这个驱动器)和别名，如图 3-7 所示。另外，可以添加关于数据库内容的注释。

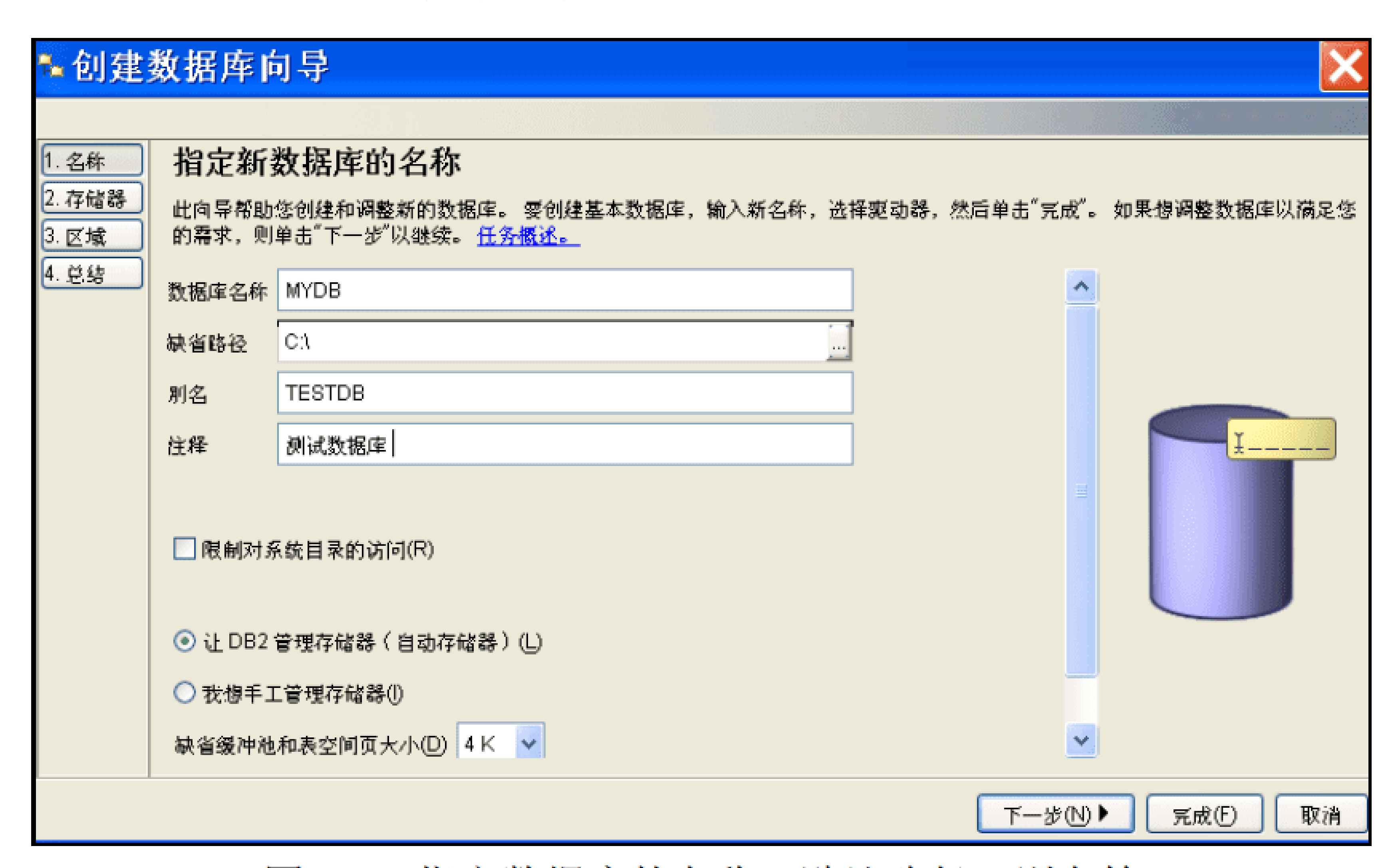


图 3-7　指定数据库的名称、默认路径、别名等

关于图 3-7 有几点需要特别注意。如果希望在数据库中使用 XML 列，那么它必须定义为 UTF-8(*Enable database for XML*)。另外，在 DB2 V9 中自动存储是数据库的默认设置。如果希望覆盖这个默认设置，就必须选择“我想手工管理存储器”选项。

创建数据库向导：用户/编目/临时表

向导的后 3 个对话框要求填写关于如何创建用户、编目和临时表空间的信息。如果选择“低维护”选项，向导就会创建 SMS 表空间。如果选择“高性能”，就需要指定用于这个表空间的设备和文件系统，如图 3-8 所示。

无论选择哪个选项，都可以指定希望分配给这个表空间的容器(文件、设备)。如果单击“添加”按钮，将会显示另一个对话框，如图 3-9 所示，可以在这里定义要使用的容器。

如果没有为表空间指定容器或文件，DB2 将在前面指定的默认驱动器上自动地生成一个。

创建数据库向导：性能选项

可以设置两个性能参数：EXTENTSIZE 和 PREFETCHSIZE，如图 3-10 所示。

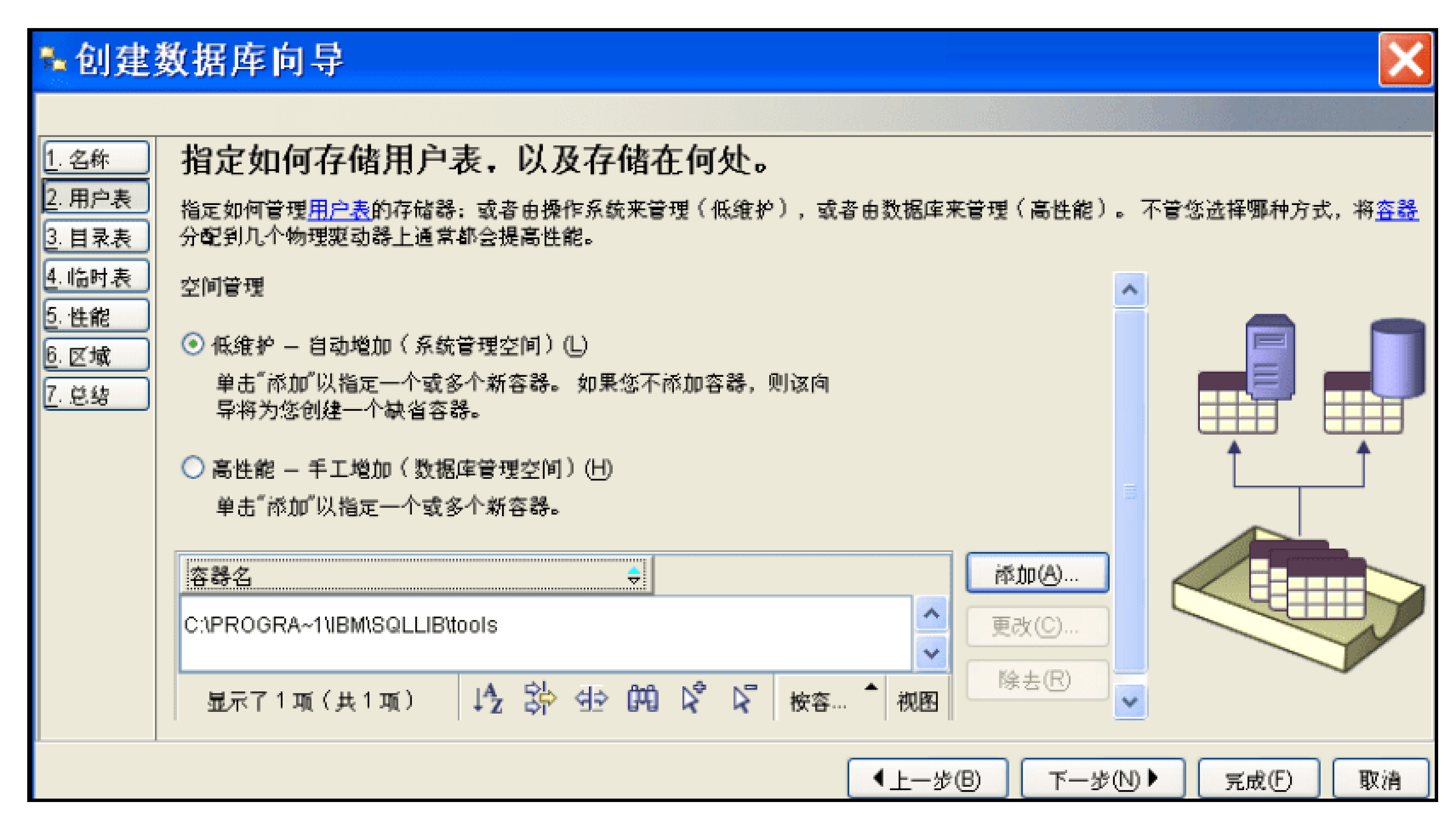


图 3-8　指定如何创建用户、编目和临时表空间

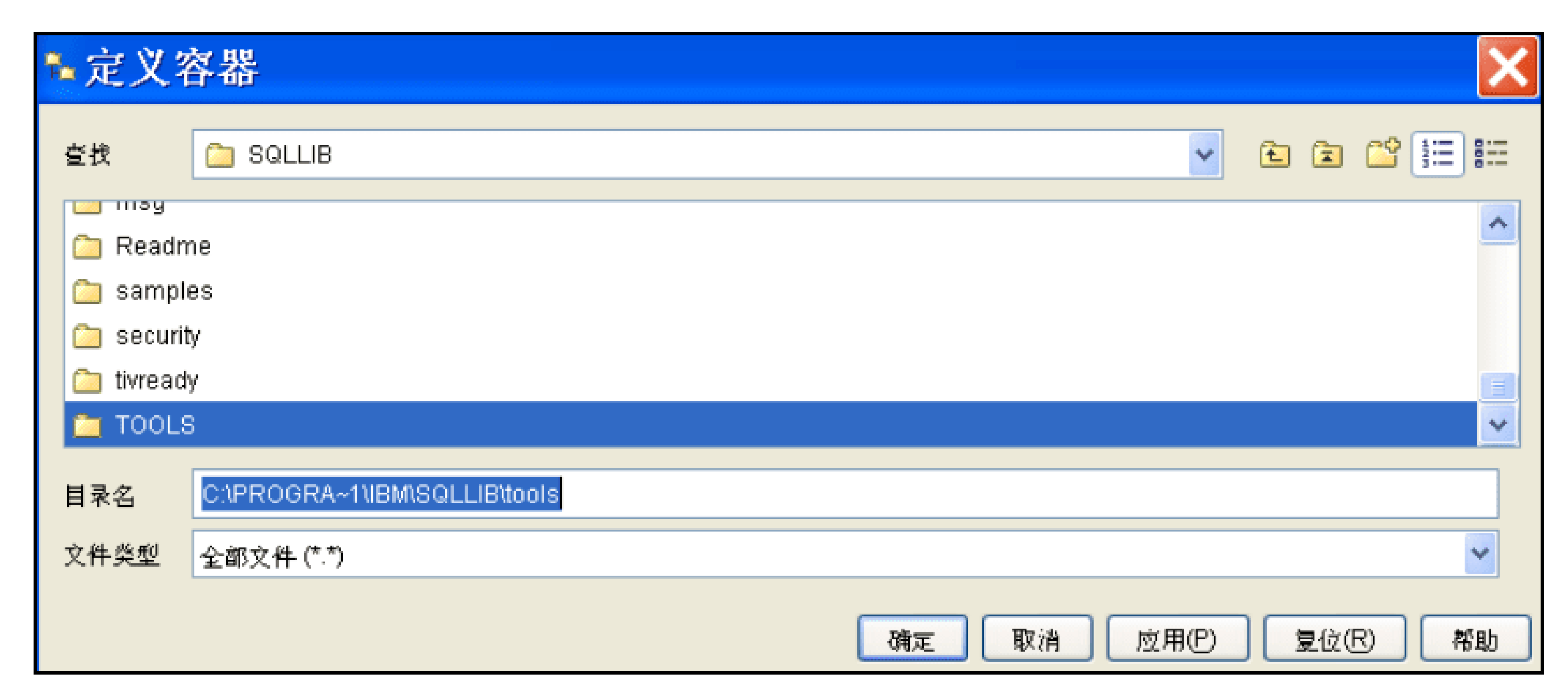


图 3-9　定义容器



图 3-10　调整数据库的性能

我们来看这两个参数的作用：

- EXTENTSIZE：扩展数据块(extent)是表空间的容器中的一个空间单元。数据库对象(除了 LOB 和 long varchar 之外)都存储在 DB2 中的页面中。这些页面组合成扩展数据块。扩展数据块大小是在表空间级定义的。一旦为表空间指定了扩展数据块大小，就不能改变了。数据库配置参数 DFT_EXTENT_SZ 指定数据库中所有表空间的默认扩展数据块大小。这个值的范围是 2 到 256 个页面；因此，绝对大小是从 8KB 到 1024KB(对于 4KB 页面)，或者从 16KB 到 2048KB(对于 8KB 页面)。可以在 CREATE TABLESPACE 语句中使用 EXTENTSIZE 参数覆盖这个数字。

  如果打算在表的设计中使用多维聚簇(MDC)，扩展数据块就是一个重要的设计决定。MDC 表将为创建的每个新的维集分配一个扩展数据块。如果扩展数据块太大，那么扩展数据块的很大一部分有可能是空的(对于包含很少记录的维集)。关于 MDC 及其对 EXTENTSIZE 的影响的更多信息，请参考《深入解析 DB2》一书。
- PREFETCHSIZE：连续的预读取是指数据库管理程序能够提前预测查询，在实际引用页面之前读取这些页面。这种异步的检索可以显著减少执行时间。可以通过修改 CREATE TABLESPACE 语句中的 PREFETCHSIZE 参数来控制执行预读取的积极程度。在默认情况下，这个值设置为 DFT_PREFETCH_SZ 数据库配置参数。这个值代表在 DB2 触发预获取请求时每次读取多少个页面。通过将这个值设置为扩展数据块大小的倍数，可以并行地读取多个扩展数据块。当表空间的容器在不同的硬盘上时，这个功能甚至效率更高。

这些参数的默认值对于许多应用程序是合适的，但是对于执行许多查询或分析大量数据的应用程序，可以考虑设置更高的 PREFETCHSIZE。

创建数据库向导：代码页和整理次序

在数据库创建过程中，遇到的下一个选项涉及代码页和整理次序，如图 3-11 所示。

当一个 DB2 应用程序绑定到 DB2 数据库时，会对应用程序和数据库的代码页进行比较。如果它们的代码页不相同，就会尝试对每个 SQL 语句执行代码页转换。如果使用与访问的数据库不同的代码页，那么一定要确保代码页是兼容的并可以执行转换。

在默认情况下，数据库的整理次序根据 CREATE DATABASE 命令中使用的编码集进行定义。如果指定选项 COLLATE USING SYSTEM，就根据为数据库指定的 TERRITORY 对数据值进行比较。如果使用选项 COLLATE USING IDENTITY，那么以逐字节的方式使用二进制表示来比较所有值。在需要以本机(二进制)格式存储数据时，要避免使用有代码页的数据类型。一般情况下，使用相同的应用程序代码页和数据库代码页是有好处的，可

以避免进行代码页转换。

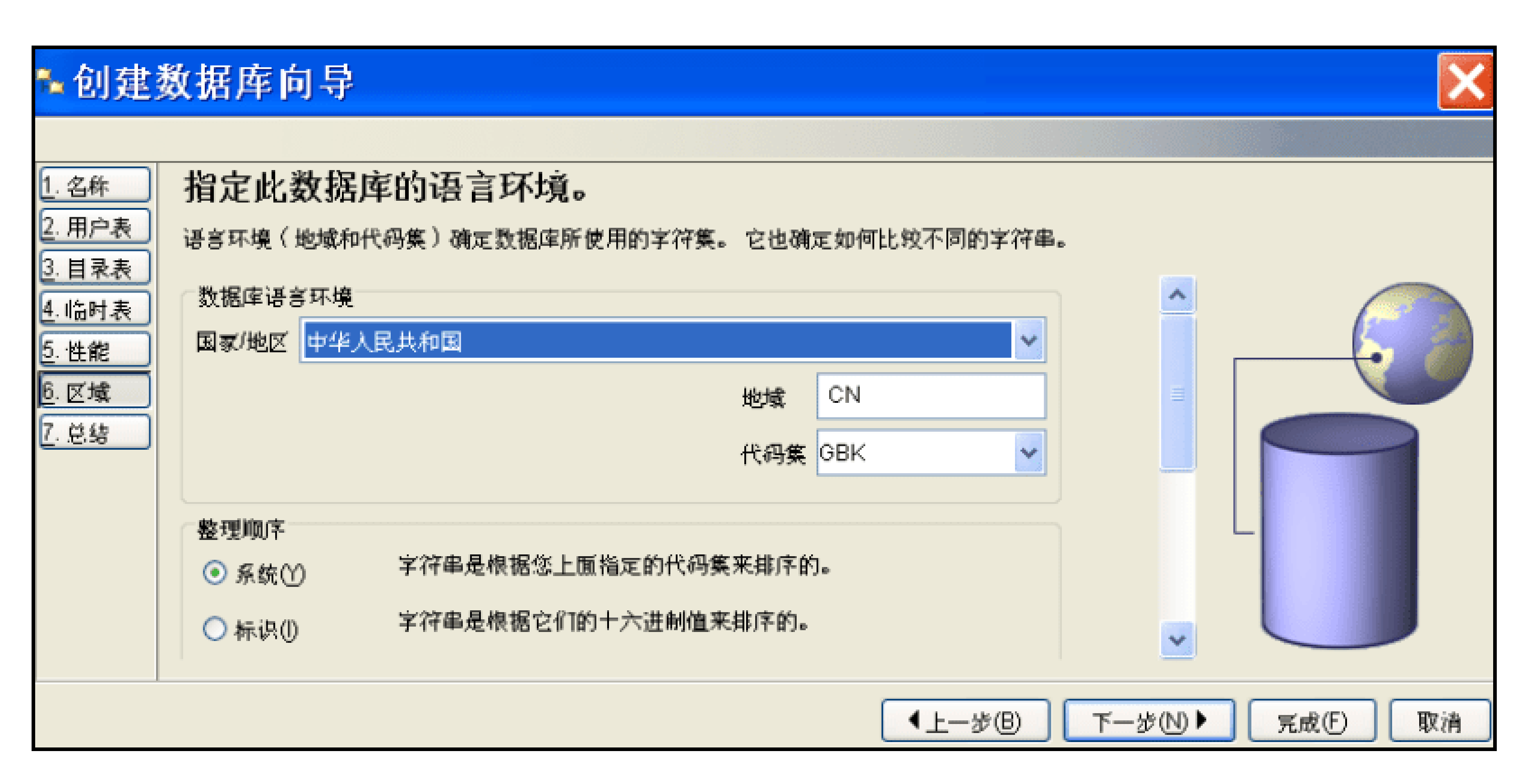


图 3-11　指定数据库的语言环境

创建数据库向导：创建总结

在设置好所有参数之后，创建数据库向导会显示一个总结页面，其中总结了您做出的所有选择，如图 3-12 所示。

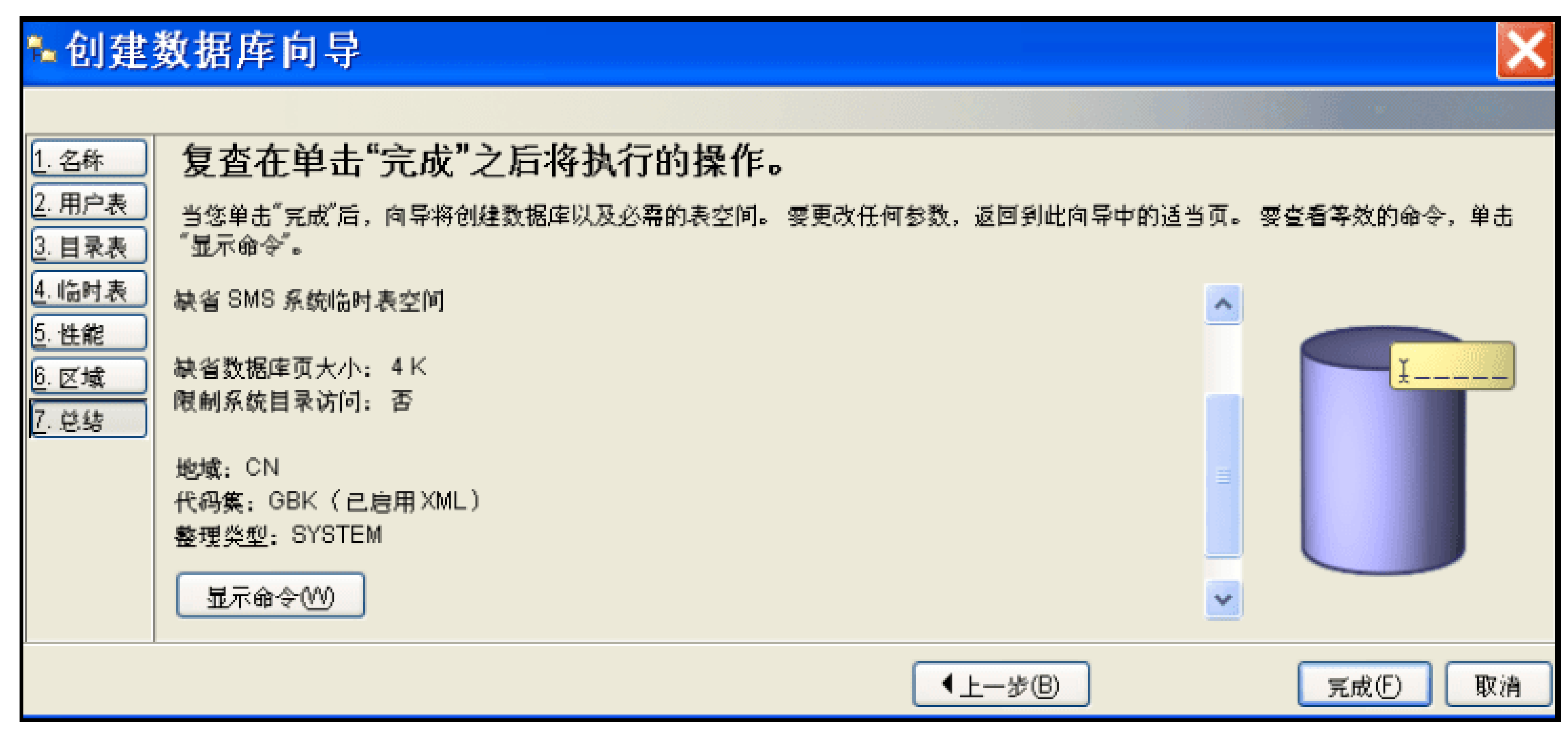


图 3-12　查看数据库创建总结

总结页面上一个极其有用的特性是“显示命令”按钮。如果单击它，就会看到用来创建数据库的 DB2 命令，如图 3-13 所示。

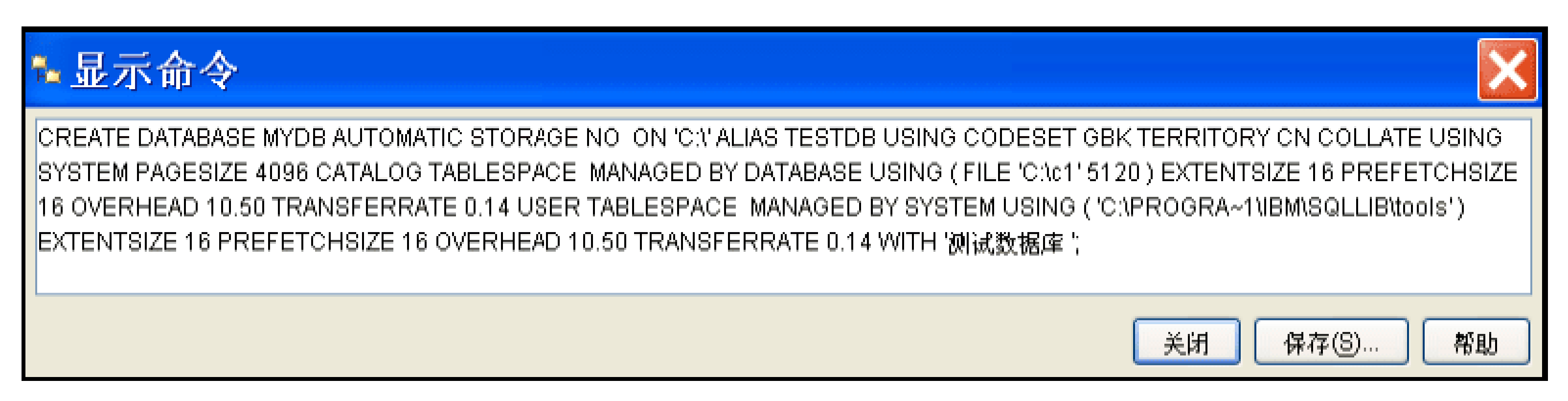


图 3-13　显示数据库的创建命令

可以保存这个命令以便在以后执行，或者将它复制并粘贴到正在开发的脚本中。如果对输入到系统中的参数满意了，就单击“完成”按钮来创建数据库。

3.1.4 数据库目录

我们在第 2 章创建实例中讲过，当我们创建一个实例时，就会生成一个实例目录。同样，当我们创建数据库时，关于该数据库的信息(包括默认信息)会存储在目录层次结构中。这就是数据库目录。此分层目录结构的创建位置取决于您在 CREATE DATABASE 命令中提供的路径信息。如果在创建数据库时未指定目录路径或驱动器的位置，那么将使用默认位置。建议我们创建数据库时明确数据库路径。数据库目录中存放的是数据库表空间、表、索引、容器等信息，这个目录至关重要，一定要注意它的安全性。

在 CREATE DATABASE 命令中指定为数据库路径的目录中，将创建一个使用实例名的子目录。这个子目录确保在同一目录下的不同实例中创建的数据库不会使用相同的路径。在实例名子目录下面，将创建一个名为 NODE0000 的子目录。这个子目录可以区分逻辑分区数据库环境中的数据库分区。在节点名目录下面，将创建一个名为 SQL00001 的子目录。此子目录的名称使用了数据库标记并表示正在创建的数据库。SQL00001 包含与第一个创建的数据库以及随后创建的具有更高编号(SQL00002 等)的数据库相关联的对象。这些子目录可以区分在 CREATE DATABASE 命令中指定的目录下的实例中创建的数据库。

目录结构如下所示：

```
<your_database_path>/<your_instance>/NODE0000/SQL00001/
```

详细的信息如图 3-14 所示。

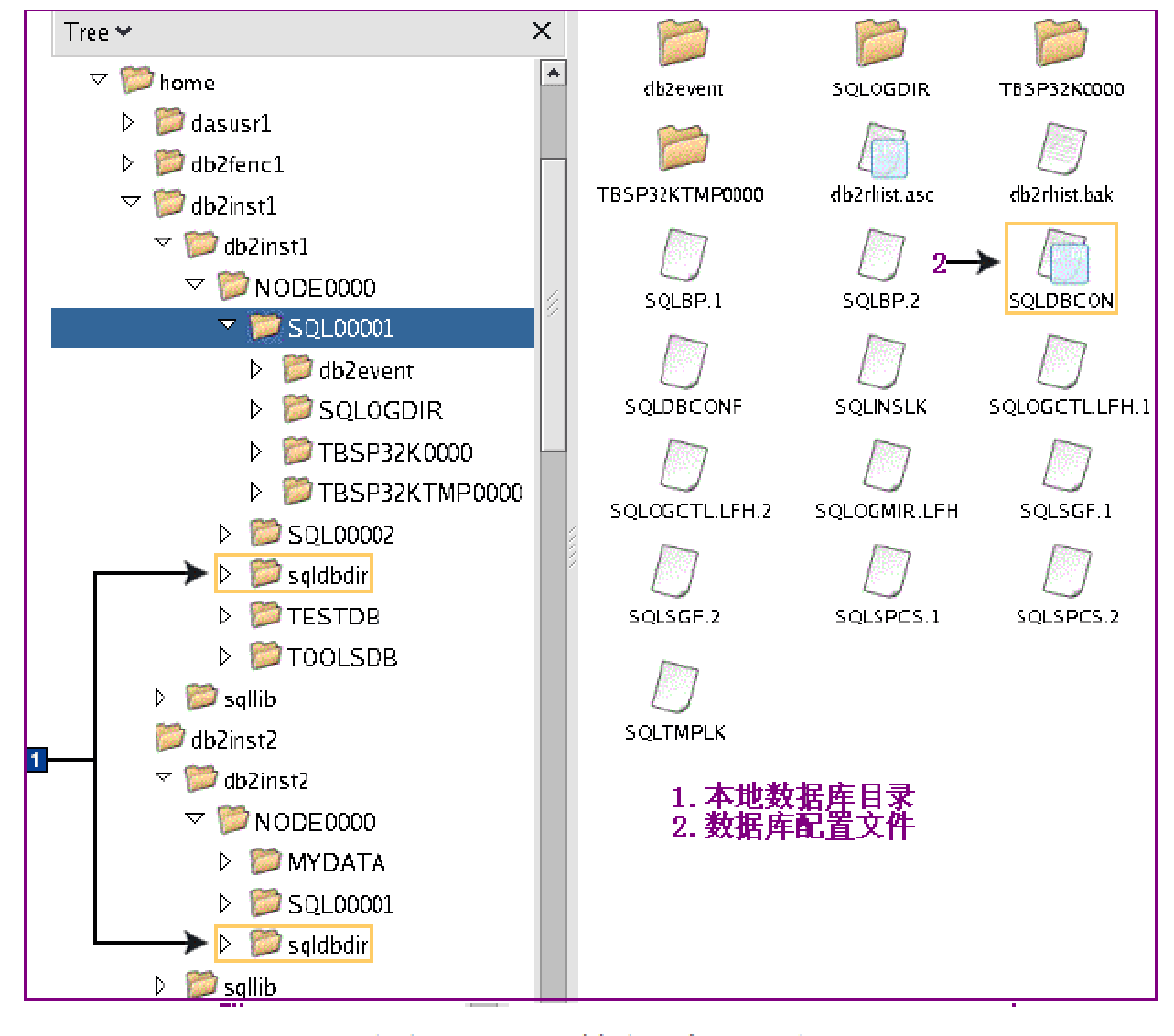


图 3-14　数据库目录

数据库目录中包含下列作为 CREATE DATABASE 命令的一部分进行创建的文件：

- 文件 SQLBP.1 和 SQLBP.2 中包含缓冲池信息。这两个文件互为副本以实现备份。
- SQLSPCS.1 和 SQLSPCS.2 文件中包含表空间信息。这两个文件互为副本以实现备份。
- 文件 SQLSGF.1 和 SQLSGF.2 包含与数据库的自动存储器有关的存储路径信息。这两个文件互为副本以实现备份。
- SQLDBCONF 文件中包含数据库配置信息。切勿编辑此文件。

注意：

SQLDBCON 文件在先前发行版中使用，并且包含在 SQLDBCONF 损坏时可以使用的类似信息。

要更改配置参数，请使用 UPDATE DATABASE CONFIGURATION 和 RESET DATABASE CONFIGURATION 语句。

- DB2RHIST.ASC 历史记录文件及其备份 DB2RHIST.BAK 中包含关于备份、复原、表装入、表重组、表空间改变和其他数据库更改的历史记录信息。

  DB2TSCHG.HIS 文件中包含日志文件级别的表空间更改的历史记录。对于每个日志文件，DB2TSCHG.HIS 中包含有助于标识日志文件影响哪些表空间的信息。表空间恢复使用此文件中的信息来确定在进行表空间恢复期间要处理哪些日志文件。可以在文本编辑器中检查这两个历史记录文件中的内容。
- 日志控制文件 SQLOGCTL.LFH.1 及其镜像副本 SQLOGCTL.LFH.2 和 SQLOGMIR.LFH 中包含有关活动日志的信息。崩溃恢复处理过程使用这些文件中的信息来确定要在日志中后退多远来开始崩溃恢复。SQLOGDIR 子目录包含实际的日志文件。

注意：

您应确保不要将日志子目录映射到用于存储数据的磁盘。这样，在磁盘发生问题时，只会影响到数据或日志，而不会同时影响这两者。由于日志文件与数据库容器不会争用同一磁盘磁头的移动，因此这可提供很多性能方面的好处。要更改日志子目录的位置，请更改 *newlogpath* 数据库配置参数。这部分内容会在第 7 章讲解。

- SQLINSLK 文件用于确保一个数据库只能由数据库管理器的一个实例使用。
- 在创建数据库的同时，还创建了详细死锁事件监视器。详细的死锁事件监视器文件存储在目录节点的数据库目录中，叫 db2detaildeadlock。

非自动存储器数据库中的 SMS 数据库目录的其他信息

在非自动存储器数据库中，SQLT*子目录包含运作数据库所需的默认“系统管理的空间”(SMS)表空间。创建数据库时会生成 3 个默认表空间：

- SQLT0000.0 子目录中包含带有系统目录表的目录表空间；
- SQLT0001.0 子目录中包含默认临时表空间；
- SQLT0002.0 子目录中包含默认用户数据表空间。

每个子目录或容器中都会创建一个名为 SQLTAG.NAM 的文件。这个文件可以标记正在使用中的子目录，因此在以后创建其他表空间时，不会尝试使用这些子目录。

此外，名为 SQL*.DAT 的文件中还存储有关子目录或容器包含的每个表的信息。星号(*)将被唯一的一组数字取代，用来识别每个表。对于每个 SQL*.DAT 文件，可能有一个或多个下列文件，这取决于表类型、表的重组状态或者表是否存在索引、LOB 或 LONG 字段：

- SQL*.BKM(如果是 MDC 表，那么包含块分配信息)
- SQL*.LF(包含 LONG VARCHAR 或 LONG VARGRAPHIC 数据)
- SQL*.LB(包含 BLOB、CLOB 或 DBCLOB 数据)
- SQL*.XDA(包含 XML 数据)
- SQL*.LBA(包含关于 SQL*.LB 文件的分配和可用空间信息)
- SQL*.INX(包含索引表数据)
- SQL*.IN1(包含索引表数据)
- SQL*.DTR(包含用于重组 SQL*.DAT 文件的临时数据)
- SQL*.LFR(包含用于重组 SQL*.LF 文件的临时数据)
- SQL*.RLB(包含用于重组 SQL*.LB 文件的临时数据)
- SQL*.RBA(包含用于重组 SQL*.LBA 文件的临时数据)

如果我们创建了多个数据库，可以通过 db2 list db directory on dbpath 查看每一个数据库的目录。

数据库目录对于应用和数据库用户来说是透明的，他们看到的是数据库逻辑层面的表、索引等对象。而数据库目录是面向 DBA 的，所以 DBA 了解数据库的物理存储模型和逻辑存储模型。逻辑模型和物理模型是用系统编目表来统一的。

3.2　表空间设计

3.2.1　创建表空间

表空间建立数据库系统使用的物理存储设备与用来存储数据的逻辑对象或表之间的关系。我们在前面创建数据库部分讲解了表空间的类型，对于非自动存储器表空间，在创建表空间时，必须知道将引用的容器的设备名或文件名。另外，必须知道与要分配给表空间的每个设备名或文件名及分配空间大小。对于自动存储器表空间，数据库管理器将根据与数据库关联的存储路径将容器指定给表空间。

在一个数据库内创建表空间，会将容器分配到表空间，并在数据库系统目录表中记录它的定义和属性，然后就可以在此表空间内创建表。当创建数据库时，会创建 3 个初始表空间。这 3 个初始表空间的页大小基于使用 CREATE DATABASE 命令时建立或接受的默认值。此默认值还表示所有将来 CREATE BUFFERPOOL 和 CREATE TABLESPACE 语句的默认页大小。如果在创建数据库时不指定页大小，那么默认页大小是 4KB。如果在创建表空间时不指定页大小，那么默认页大小是创建数据库时设置的页大小。

创建表空间可以通过控制中心或命令行创建。使用控制中心创建表空间如图 3-15 所示。

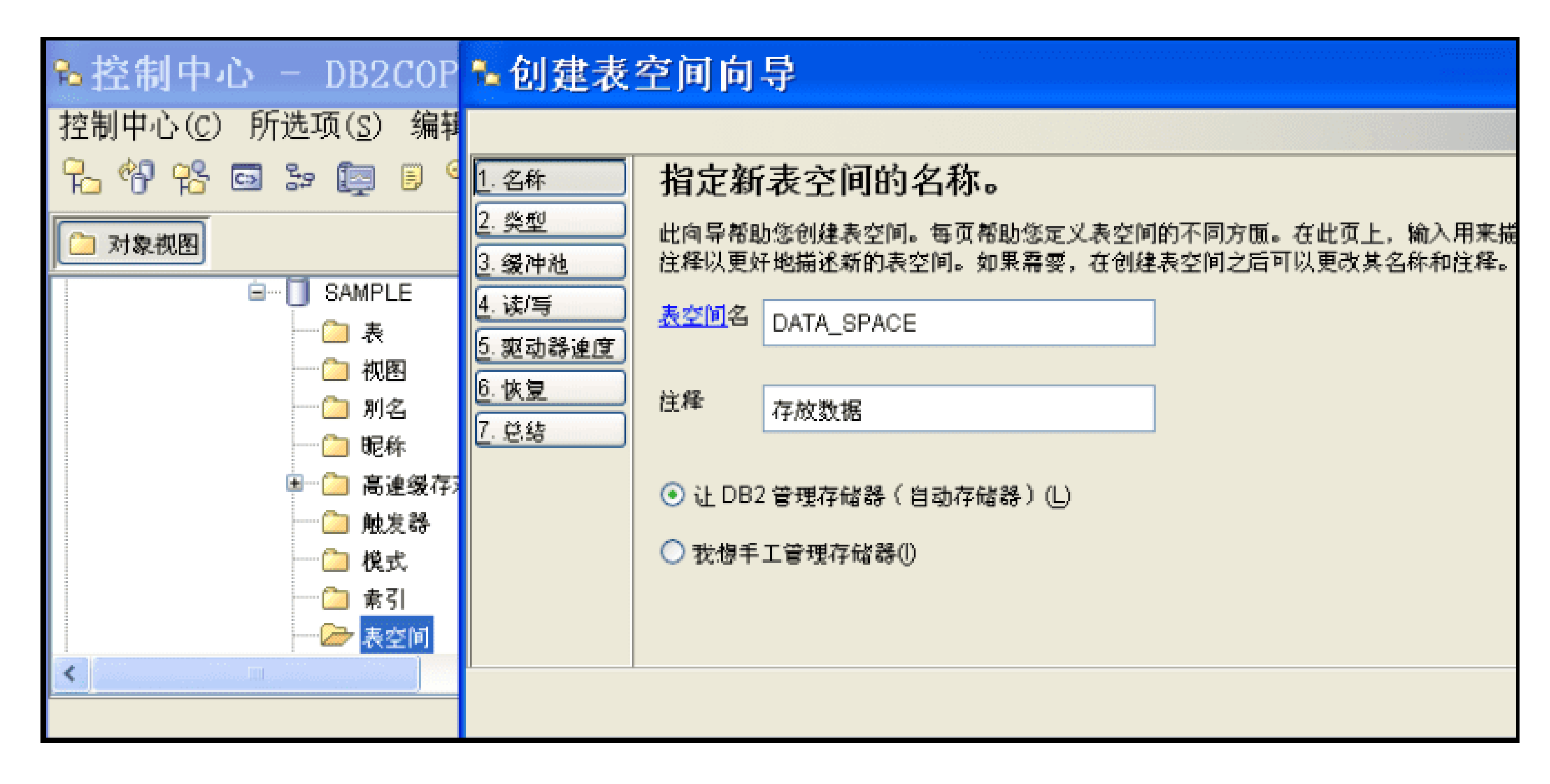


图 3-15　使用控制中心创建表空间

用图形化界面创建表空间比较简单，下面我们重点讲解如何使用命令行创建表空间。

1. 创建用户表空间

要使用命令行来创建 SMS 表空间，请输入：

```
CREATE TABLESPACE <NAME> MANAGED BY SYSTEM   USING ('<path>')
```

要使用命令行来创建 DMS 表空间，请输入：

```
    CREATE TABLESPACE <NAME>   MANAGED BY DATABASE   USING (DEVICE |FILE'<path>'
<size>)
```

要使用命令行来创建自动存储器表空间，请输入下列任一语句：

```
    CREATE TABLESPACE <NAME>
    CREATE TABLESPACE <NAME> MANAGED BY AUTOMATIC STORAGE
```

通过使用 3 个不同的驱动器上的 3 个目录，下列 SQL 语句在 Window 上创建了一个 SMS 表空间：

```
    CREATE TABLESPACE TS1 MANAGED BY SYSTEM USING ('d:\nxz_tbsp', 'e:\nxz_tbsp',
'f:\nxz_tbsp')
```

以下 SQL 语句使用各自有 5000 页的两个文件容器创建了一个 DMS 表空间：

```
    CREATE TABLESPACE TS2 MANAGED BY DATABASE
    USING (FILE'd:\db2data\acc_tbsp' 5000, FILE'e:\db2data\acc_tbsp' 5000)
```

注意在创建 DMS 表空间时，表空间文件容器不需要创建，DB2 自动来创建(裸设备容器无法自动创建，需要 root 用户参与)。

在前面两个示例中，为表空间容器提供了显式的名称。但是，如果指定相对容器名，那么将在为该数据库创建的子目录中创建容器。

在创建表空间容器时，数据库管理器会创建任何不存在的目录和文件。例如，如果将容器指定为/prod/user_data/container1，而目录/prod 不存在，那么数据库管理器会创建目录/prod 和/prod/user_data。

在 Linux/UNIX 上，数据库管理器创建的任何目录都是使用权限位 700 创建的。这意味着只有实例所有者才拥有读写访问权和执行访问权。因为只有实例所有者具有这种访问权，所以当正在创建多个实例时，可能会出现下列情况：

- 使用与上面描述的相同的目录结构，假定目录级别/prod/user_data 不存在。
- user1 创建一个实例(默认情况下命名为 user1)，接着创建一个数据库，然后创建一个表空间，且/prod/user_data/container1 作为该表空间的一个容器。
- user2 创建一个实例(默认情况下命名为 user2)，接着创建一个数据库，然后尝试创建一个表空间，且/prod/user_data/container2 作为该表空间的一个容器。

因为数据库管理器根据第一个请求使权限位 700 创建了目录级别/prod/user_data，所以 user2 没有对这些目录级别的访问权，因此不能在这些目录中创建 container2。在这种情况下，CREATE TABLESPACE 操作将失败。

解决此冲突有两种方法：

- 在创建表空间之前创建目录/prod/user_data，并将许可权设置为 user1 和 user2 创建表空间所需的任何访问权。如果所有级别的表空间目录都存在，那么数据库管理器不会修改访问权。
- 在 user1 创建/prod/user_data/container1 之后，将/prod/user_data 的许可权设置为 user2 创建表空间所需的任何访问权。

如果数据库管理器创建了一个子目录，那么在删除该表空间时数据库管理器也可能将该子目录删除。

下列 SQL 语句在 AIX 系统上创建了一个使用具有 10 000 页的 3 个裸设备作为表空间容器的 DMS 表空间，并指定它们的 I/O 特征。

```
    CREATE TABLESPACE TS1  MANAGED BY DATABASE
     USING (DEVICE '/dev/rdblv6' 10000, DEVICE '/dev/rdblv7' 10000, DEVICE
'/dev/rdblv8' 10000) OVERHEAD 7.5  TRANSFERRATE 0.06
```

在此 SQL 语句中提到的裸设备必须已经存在，且实例所有者和 SYSADM 组必须能够写入这些设备。

您还可以创建一个表空间，它使用的页大小比默认的 4KB 更大。下列 SQL 语句在 Linux 和 UNIX 系统上创建一个具有 8KB 页大小的 SMS 表空间。

```
    CREATE TABLESPACE SMS8K  PAGESIZE 8192  MANAGED BY SYSTEM
             USING ('FSMS_8K_1')   BUFFERPOOL BUFFPOOL8K
```

注意相关联的缓冲池也必须具有相同的 8KB 页大小。而且只有在激活了创建的表空间所引用的缓冲池之后才能使用该表空间。

2. 创建系统临时表空间

系统临时表空间用来存储分组、排序、连接、重组、创建索引操作等中间结果。数据库必须始终至少有一个这样的表空间。创建数据库时，定义的 3 个默认表空间之一便是名为“TEMPSPACE1”的系统临时表空间。

要创建另一个系统临时表空间，可使用 CREATE TABLESPACE 语句。例如：

```
    CREATE SYSTEM TEMPORARY TABLESPACE tmp_tbsp
    MANAGED BY SYSTEM     USING ('d:\tmp_tbsp','e:\tmp_tbsp')
```

对于每个页大小至少应具有一个和该页大小匹配的系统临时表空间。

3. 创建用户临时表空间

用户临时表空间不是在创建数据库时默认创建的。如果您的应用程序需要使用临时表，您需要创建将驻留临时表的用户临时表空间。用户临时表空间通常用来批量插入、批量更新和批量删除以加快速度。

使用 DECLARE GLOBAL TEMPORARY TABLE 语句声明临时表时必须要求用户临时表空间存在。

要创建用户临时表空间，可使用 CREATE TABLESPACE 语句：

```
CREATE USER TEMPORARY TABLESPACE usr_tbsp  MANAGED BY DATABASE
USING (FILE 'd:\db2data\user_tbsp' 5000,  FILE 'e:\db2data\user_tbsp' 5000)
```

3.2.2 表空间维护

1. 查看表空间

可以使用 DB2 LIST TABLESPACES[SHOW DETAIL]来查看表空间的详细信息。

LAST TABLESPACES 命令的输出信息如下：

```
                    Tablespaces for Current Database
Tablespace ID                       = 0
Name                                = SYSCATSPACE
Type                                = System managed space
Contents                            = Any data
State                               = 0x0000
  Detailed explanation:    Normal
Tablespace ID                       = 1
Name                                = TEMPSPACE1
Type                                = System managed space
Contents                            = System Temporary data
State                               = 0x0000
  Detailed explanation:    Normal
Tablespace ID                       = 2
Name                                = USERSPACE1
Type                                = System managed space
Contents                            = Any data
State                               = 0x0000
  Detailed explanation:    Normal
```

上面所示的这 3 个表空间是通过 CREATE DATABASE 命令自动创建的。用户可以通过在该命令中定制表空间选项来覆盖默认的表空间创建选项。但是在创建数据库时必须创建一个系统编目表空间和至少一个常规表空间，以及至少一个系统临时表空间。通过使用

CREATE DATABASE 命令或以后使用 CREATE TABLESPACE 命令，可以创建更多的所有类型的表空间(系统表空间除外)。上述 3 个表空间中，系统编目表空间和系统临时表空间都是只读的，用户不可以在上面创建用户表，如下所示。

```
C:\Program Files\IBM\SQLLIB\BIN>db2 create table t(i int) in  SYSCATSPACE
DB21034E  该命令被当作 SQL 语句来处理，因为它是无效的“命令行处理器”命令。在 SQL 处理期间，它返回：
SQL0287N  SYSCATSPACE 不可用于用户对象。 SQLSTATE=42838
```

查看表空间及容器的属性

指定 LIST TABLESPACES 命令的 SHOW DETAIL 选项将显示其他信息：

```
LIST TABLESPACES SHOW DETAIL
```

默认情况下，将列出创建数据库时所创建的那 3 个表空间。LIST TABLESPACES SHOW DETAIL 命令的输出信息如下：

```
Tablespaces for Current Database
Tablespace ID                       = 2
Name                                = USERSPACE1
Type                                = Database managed space
Contents                            = Any data
State                               = 0x0000
  Detailed explanation:    Normal
Total pages                         = 25000---------总页数
Useable pages                       = 24904--------可用页数
Used pages                          = 336--------使用页数
Free pages                          = 24568----空闲页数
High water mark (pages)             = 336
Page size (bytes)                   = 4096
Extent size (pages)                 = 32
Prefetch size (pages)               = 16
Number of containers                = 1
```

要列出容器，我们需要使用以上输出中的 Tablespace ID：

```
LIST TABLESPACE CONTAINERS FOR 2
```

查看表空间容器的情况，使用 LIST TABLESPACE CONTAINERS 命令：

```
          Tablespace Containers for Tablespace 2
 Container ID                      = 0
 Name                              = C:\DB2\NODE0000\SQL00003\SQLT0002.0
 Type                              = Path
```

该命令将列出指定表空间中的所有容器。如上所示的路径指向容器物理上所在的位置。

表空间状态

查看一个数据库中的表空间的状态，可以使用命令：

```
list tablespaces show detail
```

一个表空间可以有多种不同的状态，如下所示：

0x0	Normal
0x1	Quiesced：SHARE
0x2	Quiesced：UPDATE
0x4	Quiesced：EXCLUSIVE
0x8	Load pending
0x10	Delete pending
0x20	Backup pending
0x40	Rollforward in progress
0x80	Rollforward pending
0x100	Restore pending
0x100	Recovery pending(not used)
0x200	Disable pending
0x400	Reorg in progress
0x800	Backup in progress
0x1000	Storage must be defined
0x2000	Restore in progress
0x4000	Offline and not accessible
0x8000	Drop pending
0x2000000	Storage may be defined
0x4000000	StorDef is in 'final' state
0x8000000	StorDef was changed prior to rollforward
0x10000000	DMS rebalancer is active
0x20000000	TBS deletion in progress
0x40000000	TBS creation in progress
0x8	For service use only

关于表空间状态的详细信息，请读者参考“第 15 章：DB2 常见问题总结”中内容。

2. 修改表空间

可使用控制中心或命令行来改变表空间。要使用命令行来改变表空间，可使用 ALTER TABLESPACE 语句。可以改变 SMS、DMS 和自动存储器容器，还可以重命名表空间，并将它从脱机方式切换至联机方式。

对 SMS 表空间，我们只能增加容器，对于 DMS 表空间，可以添加、扩展、重新平衡、删除或减少容器，或者调整容器大小。我们重点讲解 DMS 表空间的修改。使用控制中心修改表空间如图 3-16 所示。

下面我们重点讲解如何使用命令行来修改表空间。

图 3-16　使用控制中心修改表空间

添加或扩展 DMS 容器

通过将一个或多个容器添加至 DMS 表空间(即使用 MANAGED BY DATABASE 子句创建的表空间)，可以增大该表空间的大小。

当将新容器添加到表空间或扩展现有容器时，可能会发生表空间重新平衡(rebanlance)。重新平衡过程涉及将表空间扩展数据块从一个位置移至另一位置。在此过程中，将尝试在表空间内分割数据。重新平衡不必在所有容器上进行，但这取决于许多因素，例如现有容器配置、新容器的大小和表空间满的程度。

在重新平衡期间，不限制对该表空间的访问。如果需要添加多个容器，那么应该同时添加这些容器，以减少重新平衡的次数。虽然重新平衡期间表空间仍然可以访问，但是我们还是尽量避免在业务高峰期间增加容器，因为数据重新平衡期间系统上有很大的 I/O 活动。关于表空间重新平衡其实 DB2 还有一些高级选项，但这部分内容超出了本书讲解范围。如果读者感兴趣可以参见《深入解析 DB2》一书。

要使用命令行将容器添加到 DMS 表空间，请输入以下内容：

```
ALTER TABLESPACE <name>  ADD (DEVICE '<path>' <size>, FILE '<filename>' <size>)
```

以下示例说明如何将两个新设备容器(各含 10 000 页)添加到 Linux 和 UNIX 系统上的表空间：

```
ALTER TABLESPACE TS1  ADD (DEVICE '/dev/rhd9'  10000,  DEVICE  '/dev/rhd10' 10000)
```

添加容器会涉及到表空间容器的重新平衡，如果您不想这样，可以使用表空间扩展来

修改容器大小，因为 extend 不会重新平衡表空间数据。

以下示例说明如何将所有容器扩展 10 000 页(各含 10 000 页)添加到 Linux 和 UNIX 系统上的表空间：

```
ALTER TABLESPACE TS1  EXTEND (ALL,10000)
```

调整 DMS 容器的大小

不能手动调用自动存储器表空间中容器的大小，否则将报错，如下所示：

```
C:\>db2 alter tablespace userspace1 extend (all 20)
DB21034E  该命令被当作 SQL 语句来处理，因为它是无效的“命令行处理器”命令。在 SQL 处
理期间，它返回：SQL20318N  不能使用"EXTEND"操作改变类型为"AUTOMATIC STORAGE"的表空间
"USERSPACE1"。 SQLSTATE=42858
```

只能将每个操作系统裸设备用作一个容器。创建了裸设备之后，其大小是固定的。当您考虑使用调整大小或扩展选项来增大裸设备容器时，应先用操作系统命令检查裸设备大小以确保您使用 ALTER TABLESPACE 命令并未将裸设备容器大小增大到大于裸设备大小。

要缩小现有容器的大小，可使用 RESIZE 选项或 REDUCE 选项。使用 RESIZE 选项时，作为语句的一部分列示的所有容器都必须增大大小或减小大小。不能在同一语句中增大某些容器而缩小其他容器。如果知道容器大小的新下限，应考虑调整大小方法。如果不知道(或不关心)容器的当前大小，那么应该考虑缩小方法。

要使用命令行来缩小 DMS 表空间中一个或多个容器的大小，请输入：

```
ALTER TABLESPACE <name>     REDUCE (FILE '<filename>' <size>)
```

以下示例说明如何在 Windows 系统上的表空间中缩小文件容器(原来为 1000 页)：

```
ALTER TABLESPACE PAYROLL    REDUCE  (FILE 'd:\hldr\finance'  200)
```

在此操作之后，文件大小就从 1000 页减少至 800 页。

要使用命令行来增大 DMS 表空间中一个或多个容器的大小，请输入：

```
  ALTER TABLESPACE <name>          RESIZE (DEVICE '<path>' <size>)
```

以下示例说明如何在 Linux 和 UNIX 系统上的表空间中增大两个设备容器(原来大小为 1000 页)：

```
ALTER TABLESPACE HISTORY  RESIZE (DEVICE '/dev/rhd7'  2000,  DEVICE
'/dev/rhd8' 2000)
```

在此操作之后，两个设备的大小都从 1000 页增加至 2000 页。

要使用命令行来扩展 DMS 表空间中一个或多个容器，请输入：

```
    ALTER TABLESPACE <name>      EXTEND (FILE '<filename>' <size>)
```

以下示例说明如何在 Windows 系统上的表空间中增大文件容器(原来大小为 1000 页)：

```
   ALTER TABLESPACE PERSNEL  EXTEND (FILE 'e:\wrkhist1'  200     FILE
'f:\wrkhist2'  200)
```

在此操作之后，两个文件的大小都从 1000 页增大至 1200 页。

删除或减少 DMS 容器

对于 DMS 表空间，可以使用 ALTER TABLESPACE 语句从表空间中删除容器或缩小容器的大小。要缩小容器，在 ALTER TABLESPACE 语句上使用 REDUCE 或 RESIZE 选项。要删除容器，在 ALTER TABLESPACE 语句上使用 DROP 选项。

仅当正在删除或缩小其大小的扩展数据块数目小于或等于表空间中“高水位标记”之上的可用数据块数目时，才允许删除现有表空间容器以及缩小现有容器的大小。高水位标记是表空间中分配的最高页的页数。此标记与表空间中已使用的页的数目不同，高水位标记下的某些扩展数据块可能可供复用。

表空间中高水位标记之上的可用扩展数据块数非常重要，原因是直至高水位标记(包括高水位标记)的所有扩展数据块必须位于表空间内的同一逻辑位置。结果表空间必须有足够的空间才能容纳所有数据。如果没有足够的可用空间，那么会产生一条错误消息(SQL20170N 或 SQLSTATE 57059)。

要删除容器，可在 ALTER TABLESPACE 语句上使用 DROP 选项。例如：

```
   ALTER TABLESPACE TS1 DROP (FILE 'file1', DEVICE '/dev/rdisk1')
```

改变自动存储器表空间

对于自动存储器表空间，不能手动调整自动存储器表空间的大小，数据库管理器将在需要时自动调整容器大小。

3. 重命名表空间

可以使用 RENAME TABLESPACE 语句来重命名表空间。不能重命名 SYSCATSPACE 表空间。不能重命名处于“前滚暂挂”或“正在前滚”状态的表空间。

可以给予现有表空间新名称，而无需关心该表空间中的个别对象。重命名表空间时，将更改所有引用该表空间的目录记录。例如：

```
   RENAME TABLESPACE TS1 TO TS2
```

注意：

当复原在备份后已被重命名的表空间时，必须在 RESTORE DATABASE 命令中使用新的表空间名。如果使用先前的表空间名，那么将找不到该名称。同样，如果使用 ROLLFORWARD DATABASE 命令前滚该表空间，也需确保使用新名称。如果使用先前的表空间名，那么将找不到该名称。

4. 将表空间从脱机状态切换至联机状态

如果与表空间相关的容器不可访问，这时表空间处于 OFFLINE 状态，要使用命令行从表空间中除去 OFFLINE 状态，请输入：

```
ALTER TABLESPACE <name>  SWITCH ONLINE
```

什么情况下会处于 OFFLINE 状态呢？我举一个实际生产中的例子。在一个双机热备 HA 的环境中，客户在主机上重新创建了使用裸设备的表空间后，未同步 HA 环境，结果导致主机故障切换到备机时，由于裸设备权限不正确而导致表空间处于 OFFLINE 状态。

5. 删除表空间

当删除表空间时，也会删除该表空间中的所有数据，释放容器，除去目录条目，并导致该表空间中定义的所有对象都被删除或标记为无效。可以通过删除表空间来重用空表空间中的容器，但是，在试图重用这些容器之前，必须落实 DROP TABLESPACE 语句。

删除用户表空间

可删除一个包含所有表数据的用户表空间，包括在该单个用户表空间中的索引和 LOB 数据。也可删除所包含的表跨几个表空间的用户表空间。即，可能表数据在一个表空间，索引在另一个表空间且任何 LOB 数据在第 3 个表空间。必须在一条语句中同时删除所有 3 个表空间。包含跨越的表的所有表空间必须全部纳入此单条语句中，否则该删除请求将失败。例如创建表的定义如下：

```
    create table xinzhuang pic(picno  int，pic clob(1g)) in data space index in
index_space long in lob_space
```

只能同时删除 3 个表空间：

```
DROP TABLESPACE DATA_SPACE，INDEX_SPACE，LOB_SPACE
```

删除用户临时表空间

仅当用户临时表空间中当前未定义已声明临时表时，才能删除该表空间。当删除表空

间时，不会尝试删除该表空间中的所有已声明临时表。

注意：
已声明临时表是在说明它的应用程序与数据库断开连接时隐式删除的。

删除系统临时表空间

如果不首先创建另一系统临时表空间，那么不能删除页大小为 4KB 的默认系统临时表空间。新的系统临时表空间必须具有 4KB 页大小，原因是数据库必须始终存在至少一个具有 4KB 页大小的系统临时表空间。例如，如果具有页大小为 4KB 的单个系统临时表空间，并且您想要将一个容器添加到该表空间(它是 SMS 表空间)，那么您必须首先添加一个具有适当数目的容器的新 4KB 页大小的系统临时表空间，然后删除旧的系统临时表空间(如果正在使用 DMS，那么可以添加容器而不必删除并重新创建表空间)。

默认系统临时表空间页大小是创建数据库时使用的页大小(默认情况下为 4KB)，但也可以为 8KB、16KB 或 32KB。

下面是用来创建系统临时表空间的语句：

```
    CREATE SYSTEM TEMPORARY TABLESPACE <name>  MANAGED BY SYSTEM USING
('<directories>')
```

创建之后，要使用命令行删除系统表空间，请输入：

```
    DROP TABLESPACE <name>
```

以下 SQL 语句创建一个称为 TEMPSPACE2 的新的系统临时表空间：

```
    CREATE SYSTEM TEMPORARY TABLESPACE TEMPSPACE2
           MANAGED BY SYSTEM USING ('d:\systemp2')
```

一旦创建了 TEMPSPACE2，那么就可使用以下命令删除原来的系统临时表空间 TEMPSPACE1：

```
    DROP TABLESPACE TEMPSPACE1
```

3.2.3　表空间设计注意事项

1.表空间类型选择

在确定应使用哪种类型的表空间来存储数据时，需要考虑一些问题。

SMS 表空间的优点：

- 根据需要，系统按需分配空间

- 由于不必预定义容器，所以创建表空间需要的初始工作较少

DMS 表空间的优点：

- 通过使用 ALTER TABLESPACE 语句，可添加或扩展容器来增加表空间的大小。现有数据可以自动在新的容器集合中重新平衡，以保持最佳 I/O 效率
- 根据存储的数据的类型，可以把一个表长字段(LF)和大对象(LOB)数据、索引和常规表数据分割存放在多个表空间中以提高性能和空间存储容量。通过分隔表数据，可以提高性能和增加每个表存储的数据量。例如，如果您要使用 4KB 页大小的大型表空间，那么可以有一个包含 2TB 正规表数据的表、有一个包含 2TB 索引数据的单独表空间和另一个包含 2TB 长型数据的单独表空间。如果这 3 种类型的数据存储在一个表空间中，那么总空间将限制为 2TB。使用较大的页大小将允许您存储更多数据
- 对范围分区数据创建的索引可以与表数据存储在不同的表空间中
- 可控制数据在磁盘上的位置(如果操作系统允许)
- 通常，精心设计的一组 DMS 表空间的性能将优于 SMS 表空间

注意：

对于性能要求很高的应用程序，特别是涉及大量 DML 操作的应用程序，建议您使用 DMS 表空间。

其实 DMS 的优势，就是数据在物理磁盘上的连续性。SMS 使用操作系统来管理空间，虽然从逻辑上看，看似所有的文件都是连续的，但是在物理磁盘上，每次文件的增大都必须要分配新的空间。所以从操作系统看来，所谓的“分配”不过是在 inode 节点中增加一个指向页的偏移，这个页是操作系统寻找出来没有被使用的，因此从磁盘上看来，一个文件可以被切分成很多块存储在不同的地方——尽管逻辑上它们是连续的。这也就是能够动态增加 size 的 SMS 文件的致命伤。不像 DMS，分配完成之后一般不会随意增加或者减少 size，SMS 的 size 增加有时可能非常频繁，因此每个文件在物理磁盘上的存储会被划分成一个个小块。这样的话尽管在逻辑上它们的条带化还是连续的，但是从物理磁盘上看它们每个 extent 之间可能并非连续，无法使用 range prefetch 直接从磁盘上读取几个连续的 extent。

而且在这两种类型的表空间上，数据的放置也会有所不同。例如，进行高效表扫描要求扩展数据块中的页在物理上是连续的。对于 SMS，操作系统的文件系统决定了每个逻辑文件页的物理放置位置。根据文件系统上其他活动的级别以及用来确定放置位置的算法的不同，可能会连续分配这些页、也可能不会。但是，对于 DMS，因为数据库管理器直接与

磁盘打交道，所以它可以确保这些页在物理上是连续的。

通常，小型个人数据库用 SMS 表空间管理最容易。另一方面，对于不断增长的大型数据库，建议使用 SMS 表空间用作临时表空间和系统编目表空间，而使用具有多个容器的单独的 DMS 表空间用于每个表。另外，建议将长字段(LF)数据和索引存储在它们自己的表空间中。

在深刻理解上述两种表空间优缺点后，我们选择表空间要综合考虑如下因素：

表中的数据量

如果计划在一个表空间中存储许多小表，那么考虑使用 SMS 充当该表空间。对于小表，DMS 表现在 I/O 和空间管理效率方面的优点就没有那么重要。SMS 的优点(仅在需要时使用)却对小表更具吸引力。如果一个表较大或者您需要更快地访问表中的数据，应考虑具有较小扩展数据块大小的 DMS 表空间。

设计数据库时，可以考虑对每个非常大的表都使用单独的 DMS 表空间，而将所有的小表组合在单个 SMS 表空间中。这种分隔还允许您根据表空间的使用选择适当的扩展数据块大小。

表数据的类型

例如，有的表可能包含不经常使用的历史记录数据；最终用户可能愿意接受较长的响应时间，来等待对此数据执行的查询。在这种情况下，可以为历史记录表使用单独的一个表空间，并将此表空间分配给访问速率较低的较便宜的物理设备。

此外，对于某些表，数据快速以及快速响应时间是非常必要的，那么需要将这些表分配给一个快速物理设备的表空间中，这样将有助于支持这些重要的数据需要。如果可以的话，可以使用内存硬盘来存放访问最频繁的配置表、参数表等。

通过使用 DMS 表空间，还可以将表数据分发在 3 个不同的表空间中：一个存储索引数据；一个存储大对象(LOB)和长字段(LF)数据；一个存储常规表数据。这允许您选择表空间特征和支持最适合该数据的那些表空间的物理设备。例如，可能会将索引数据置于可找到的最快的设备上，这样性能可显著提高。如果将一个表分布在各个 DMS 表空间中，那么在启用表空间级备份恢复时，应考虑一起备份和复原那些表空间。SMS 表空间不支持以此方式将数据分发在所有表空间中。

管理问题

某些管理功能可以在表空间级执行，但不能在数据库或表级执行。例如，备份表空间(而不是数据库)可以帮助您更好地利用时间和资源。它允许频繁地备份带有大量更改的表空间，同时仅偶尔地备份带有少量更改的表空间。

可以复原数据库或表空间。如果不相关的表在同一个表空间中，就可以选择复原数据库一个较小的部分以降低成本。一种好方法是将相关的表存放在一个表空间中。这些表可以通过参考约束相关，也可以通过定义的其他业务约束相关。

如果需要经常删除并重新创建特定表，那么应给这样的表单独创建一个 DMS 表空间，因为删除一个 DMS 表空间比删除一个表更有效率。

2. 选择合适的数据页大小

创建表空间时，需要考虑页大小。可以使用 4KB、8KB、16KB 或 32KB 页大小。我们在选择数据页大小时需要综合考虑空间需求和业务类型(性能需求)来做出选择。

空间需求

因为 DB2 中每个页大小限定了可以存储行的最大长度和存储表空间的最大值，所以我们选择数据页大小要考虑这些。对于 4KB 数据页来说，最多可以存放的行的长度是 4005 字节(4096-91 头部；8KB 为 8192-91；依此类推)，所以首先要根据行的长度来选择数据页大小。表 3-3 列出了每种数据页大小的空间使用限制，以及不同类型的表空间的数据库和索引页大小限制。

表 3-3　表空间特定于页大小的限制

表空间类型	4KB	8KB	16KB	32KB
SMS 表空间	64GB	128GB	256GB	512GB
DMS 表空间(常规)	64GB	128GB	256GB	512GB
DMS 表空间(大型)	2TB	4TB	8TB	16TB
自动存储器表空间(常规)	64GB	128GB	256GB	512GB
自动存储器表空间(大型)	2TB	4TB	8TB	16TB
临时表空间	64GB	128GB	256GB	512GB

如果数据页大小选择不当还可能造成空间浪费。例如，如果要使用 32KB 的页面大小的常规表空间来存储平均大小为 100 字节的行，则一个 32KB 的页只能存储 100 * 255 = 25500 Byte (24.9 KB)。这意味着每 32KB 中就有大约 7KB 要浪费掉。而且 DB2 V8 之前的表空间存在着一个数据页最多存放 255 行的限制，所以建议创建表空间时，尽量创建大型表空间，大型表空间一个数据页可以存放更多的容量和行数。

业务类型

我们要根据我们的业务类型选择合适的数据页大小。通常的业务类型有 OLTP、OLAP、批处理、报表，以及这几种业务混合的类型。下面我们来介绍主要业务类型的特点。

联机事务处理(OLTP)工作负载的特征是：事务需要对数据进行随机访问，通常涉及频繁插入或更新活动和返回一小组数据的查询。通常访问是随机的，并且是访问一页或几页，一般不太可能发生预存取(prefetch)。这里顺便讲一下，其实对于性能要求很高的 OLTP 应用，我们可以考虑把一些频繁访问的配置表、参数表等建在内存盘上。在 AIX 和 Linux 上都支持创建在内存上的硬盘，这样也可以大大提高速度。但是要注意保证数据的一致性。

使用裸设备容器的 DMS 表空间在这种情况下表现得最好。请注意，在 FILE SYSTEM CACHING 关闭的情况下，将 DMS 表空间与文件容器配合使用在某种程度上相当于 DMS 裸设备容器。如果业务逻辑存在大量的随机读，那么 CREATE TABLESPACE 语句中的 EXTENTSIZE 和 PREFETCHSIZE 参数的设置对于 I/O 的效率就显得不重要。但是使用 chngpgs_thresh 配置参数设置足够数目的页清理程序很重要。

OLAP 查询工作负载的特征是：事务需要对数据进行顺序访问或部分顺序访问，并常常返回大的数据集。使用多个设备容器且每个容器都在单独的磁盘上的 DMS 表空间最有可能提供有效的并行预存取。应该将 CREATE TABLESPACE 语句中的 PREFETCHSIZE 参数的值设置为 EXTENTSIZE 参数的值乘以容器设备数之积。此外，可以将预取大小指定为 -1，此时数据库管理器将自动(automatic)选择合适的预取大小。这允许数据库管理器以并行方式从所有容器中预取。如果容器的数目更改，或需要使预取更多或更少，那么可以使用 ALTER TABLESPACE 语句相应地更改 PREFETCHSIZE 值；如果把 PREFETCHSIZE 设置为 AUTOMATIC，添加容器后，数据库会自动调整 prefetchsize 的大小，所以强烈建议把 PREFETCHSIZE 设置为 AUTOMATIC 或－1。

混合工作负载的目标是：对于 OLTP 工作负载，使单个 I/O 请求尽可能有效率；而对于查询工作负载，最大程度地提高并行 I/O 的效率。

选择表空间页大小的注意事项如下所示：

- 对于执行随机行读写操作的 OLTP 应用程序，通常最好使用较小的页大小(4KB、8KB)，这样不需要的行就不会浪费缓冲池空间；
- 对于一次访问大量连续行的决策支持系统(DSS)和 OLAP 应用程序，页大小大一些(16KB、32KB)会比较好，这样就能减少读取特定数目的行所需的 I/O 请求数。较大的页大小还允许您减少索引中的层数，因为在一页中可以保留更多的行指针；
- 越大的页，支持的行越长。应根据业务需求来选择合适的数据页；

- 在默认的 4KB 页上，一个表只能有 500 列，而更大的页大小(8KB、16 KB 和 32 KB)支持 1012 列；
- 表空间的最大大小与表空间的页大小成正比，见表 3-3。

3. 扩展数据块大小选择注意事项

EXTENTSIZE 指定在跳到下一个容器之前，可以写入到一个容器中的 PAGESIZE 页面的数量。存储数据时数据库管理器反复均衡使用所有容器。该参数只有在表空间中有多个容器时才起作用。选择合理的 EXTENTSIZE 会对表空间的性能产生重大影响。因为这个参数是在创建表空间时定义的，之后不能修改，所以我们创建时必须选择合理的 EXTENTSIZE。对于自动存储表空间来说，数据库自动选择 EXTENTSIZE 大小，对于非自动存储来说，我们选择 EXTENTSIZE 大小时必须综合考虑以下因素：

表空间的大小和类型

DMS 表空间中的空间一次分配给表一个扩展数据块。当插入该表而一个扩展数据块变满时，会分配新的扩展数据块，直到彻底用完容器为止。

将 SMS 表空间中的空间一次分配给表一个扩展数据块或者一次分配给表一页。当插入该表而一个扩展数据块或页变满时，会分配新的扩展数据块或页，直到使用了文件系统中的所有扩展数据块或页为止。当使用 SMS 表空间时，允许进行多页文件分配(注：DB2 V8 之前需要执行 db2empfa 激活多页分配，V8 以后默认自动设置了多页分配)。多页文件分配允许分配扩展数据块而不是一次分配一页。

每个表对象都是单独存储的，每个对象按需要分配新的扩展数据块。每个 DMS 表对象还与称为扩展数据块映像的元数据对象配成一对，该元数据对象描述该表空间中属于该表对象的所有扩展数据块。用于扩展数据块映像的空间也是以一次一个扩展数据块的方式分配。因此，DMS 表空间中对象的初始空间分配是两个扩展数据块(SMS 表空间中对象的初始空间分配是一页)。

如果您在一个 DMS 表空间中有多个较小的表，那么可能要分配相对大的空间来存储相对少量的数据。在这种情况下，应该指定小的扩展数据块大小。另一方面，如果您有一个增长速率高的非常大的表，且您使用具有较小扩展数据块大小的 DMS 表空间，那么可能会产生与其他扩展数据块的频繁分配相关的不需要的开销。

下面的经验法则是建立在表空间中每个表的平均大小的基础上的：

- 如果小于 50 MB，EXTENTSIZE 为 8
- 如果介于 50 到 500MB 之间，则 EXTENTSIZE 为 16
- 如果介于 500 MB 到 5GB 之间，则 EXTENTSIZE 为 32

● 如果大于 5GB，则 EXTENTSIZE 为 64

对这些表访问的类型

对于 OLAP 数据库和大部分对表的访问包括许多查询或处理大量数据的事务(仅限于查询)的表，或者增长速度很快的表，那么从表中预取数据可以显著改善性能，应使用较大的 extent。反之，对于较小的频繁更改、频繁随机读取的小的配置表，建议使用较小的 extent。

如果打算在表的设计中使用多维聚簇(MDC)，扩展数据块就是一个重要的设计决定。MDC 表将为创建的每个新的维集分配一个扩展数据块。如果扩展数据块太大，那么扩展数据块的很大一部分有可能是空的(对于包含很少记录的维集)。这会造成非常大的空间浪费。关于 MDC 及其对 EXTENTSIZE 的影响的更多信息请参见《深入解析 DB2》一书。

3.2.4　prefechsize 大小选择

当执行数据预取时每次从表空间读取的页数。预取操作在查询使用所需的数据之前读入这些数据，因为数据已经存在于内存中了，这样一来查询在使用这些数据的时候就不必等待执行 I/O 了。当数据库管理器确定顺序 I/O 是适当的，并且确定预取操作可能有助于提高性能时，它就选择预取操作。

通过使用 ALTER TABLESPACE 可以轻易地修改预取大小。最优设置差不多是下面这样的：

```
    Prefetch Size = (# Containers of the table space on different physical disks)
* Extent Size
```

如果表空间驻留在一个磁盘阵列上，则如下设置：

```
    PREFETCH SIZE = EXTENT SIZE * (# of non-parity disks in array)
```

注意：

在 DB2 V9 版本以后，可以在创建表空间的时候自动预取大小。

如果添加或删除容器后，可能忘记更新表空间的预取大小，那么应考虑允许数据库管理器自动确定预取大小。如果忘记更新预取大小，那么数据库性能可能会明显降低。所以您可以在创建表空间时指定 prefetchsize 为 automatic，这样就可以设置自动预取大小，并可以通过下面的快照监控来查看是否设置自动预取。

```
    C:\>db2 get snapshot for tablespaces on sample | more
                  表空间快照
    第一个数据库连接时间戳记                  = 2008-10-15 09:19:37.992116
    …………………………………………………………………………….
```

```
表空间扩展数据块大小(以页计)          = 4
 启用的自动预取大小 (prefetchsize)=是(automatic)
 当前在使用的缓冲池标识              = 1
 下一次启动的缓冲池标识              = 1
 使用自动存储器                      =是
 启用自动调整大小                    =是
表空间高速缓存                        = 否
表空间状态                            = 0x'00000000'
  详细解释:       正常
表空间预取大小(以页计)                = 4
略.......................
```

当然，prefetchsize 的大小设置还和 extentsize 的设置有关，所以我们首先要合理地设置 extent 的大小，然后再根据 extentsize 的大小设置 prefetchsize。一个比较好的建议是创建数据库时采用自动存储。这样由数据库管理器自动来设置 extentsize 和 prefetchsize 大小。

3.2.5 文件系统(CIO/DIO)和裸设备

我们在创建 DMS 表空间容器时可以选择使用裸设备或文件系统，下面我们来看看两者的区别。我们知道，内存的读写效率比磁盘高近万倍，因此数据库通常会在内存中开辟一片区域，称为 Buffer Pool，使数据的读写尽量在这部分内存中完成。同样地，在文件系统中，操作系统为了提高读写效率，也会为文件系统开辟一块 Buffer 用于读写数据的缓存。这样，数据库中的数据会被缓存两次。为了避免操作系统的这次缓存，我们可以采用裸设备作为数据文件的存储设备。裸设备，也称为裸分区(Raw Partiton)，是一个没有被加载(Mount)到操作系统的文件系统上的磁盘分区，它通过字符设备驱动来访问。裸设备的 I/O 读写不由操作系统控制，而是由应用程序(如数据库)直接控制。

裸设备的优点：

- 由于屏蔽了文件系统缓冲器而进行直接读写，从而具有更好的性能。对硬盘的直接读写就意味着取消了硬盘与文件系统的同步需求。这一点对于纯 OLTP 系统非常有用，因为在这种系统中，读写的随机性非常大以至于一旦数据被读写之后，它们在今后较长的一段时间内不会得到再次使用。除了 OLTP，裸设备还能够从以下几个方面改善 DSS(决策支持系统)应用程序的性能：
  - ◊ 排序：对于 DSS 环境中大量存在的排序需求，裸设备所提供的直接写功能也非常有用，因为对临时表空间的写动作速度更快。
  - ◊ 顺序访问：裸设备非常适合于顺序 I/O 动作。同样地，DSS 中常见的顺序 I/O(表/索引的全表扫描)使得裸设备更加适用于这种应用程序。

- 直接读写，不需要经过操作系统级的缓存。节约了内存资源，在一定程度上避免了内存的竞争。
- 避免了操作系统的 cache 预读功能，减少了 I/O。
- 采用裸设备避免了文件系统的开销。比如维护 I-node、空闲块等。

裸设备的缺点：

- 裸设备的空间大小管理不灵活。在放置裸设备的时候，需要预先规划好裸设备上的空间使用。还应当保留一部分裸设备以应付突发情况。这也是对空间的浪费。
- 需要操作系统 root 用户干预，因为裸设备的创建、更改权限、扩展大小等都需要 root 用户完成，增加了管理的成本。

文件系统的优点：

文件系统易于管理和维护，如文件的基本管理以及安全和备份等。

文件系统的缺点：

性能比不上裸设备。

我们在选择表空间容器时，从性能上考虑尽量采用裸设备，但是如果我们使用自动存储方式创建数据库和表空间，这种方式不支持裸设备。或者我们为了便于管理而采用文件系统方式。这时候我们采用合理设置文件系统相关选项和表空间的相关选项。下面我们来讲解文件系统方面应该注意的事项。

CIO/DIO

直接 I/O(DIO)由于可以绕过在文件系统级别进行高速缓存，从而改进内存性能。此过程可减少 CPU 开销并使得更多的内存可用于数据库实例。并发 I/O(CIO)具有 DIO 的优点，并且还可以消除串行化写访问权。与使用文件系统缓冲 I/O 相比，在具有大量事务处理工作负载和回滚时 CIO/DIO 机制可增大吞吐量。

DIO 和 CIO 在 HP-UX、Solaris、Linux 和 Windows 操作系统最新版本上都受支持。

关键字 NO FILE SYSTEM CACHING 和 FILE SYSTEM CACHING 是 CREATE 和 ALTER TABLESPACESQL 语句的一部分，允许您指定将对每个表空间使用 DIO 还是 CIO。当 NO FILE SYSTEM CACHING 有效时，只要可能，数据库管理器都会尝试使用“并发 I/O”。在不支持 CIO 的情况下(例如，当使用了 JFS 时)，将取而代之使用 DIO。

建议在表空间级别启用或禁用 UNIX、Linux 和 Windows 上的非缓冲 I/O。这将允许您在特定表空间上启用或禁用非缓冲 I/O，同时避免数据库的物理布局中的任何依赖性。它还允许数据库管理器确定每个文件最适合使用哪种 I/O，缓冲的还是非缓冲的。

NO FILE SYSTEM CACHING 子句用于启用非缓冲 I/O，从而禁用特定表空间的文件高速缓存。一旦启用了非缓冲 I/O，数据库管理器就会根据平台自动确定将使用直接 I/O 还

是并发 I/O。由于使用 CIO 可以提高性能，所以只要支持 CIO，数据库管理器就会使用它。

FILE SYSTEM CACHING 选项并不是总没有好处，例如当一个应用程序检索 LOB 或 LONG 数据时，这些大对象数据不能经过数据库缓冲池，每次应用程序需要其中一个页时，数据库管理器必须从磁盘对其进行直接读取。但是，如果 LOB 或 LONG 数据存储在 SMS 或 DMS 文件容器中，文件系统高速缓存可提供缓冲，因此也就改善了性能。

未在 CREATE TABLESPACE 语句或 CREATE DATABASE 命令中指定此属性时，数据库管理器将使用基于平台和文件系统类型的默认行为处理请求。查看是否启用 FILE SYSTEM CACHING 属性，可以使用：

- GET SNAPSHOT FOR TABLESPACES 命令(此命令会在第 12 章讲解)

例如，以下是 DB2 GET SNAPSHOT FOR TABLEPSACES ON SAPMPLE 输出：

```
表空间名                                  = USERSPACE1
   表标识                                 = 2
   表空间类型                             =数据库管理的空间
   表空间内容类型                         = 所有永久数据。大型表空间。
   表空间页大小(以字节计)                 = 4096
   表空间扩展数据块大小(以页计)           = 32
    已启用自动预取大小                     = Yes
    当前正在使用的缓冲池标识               = 1
    下一次启动的缓冲池标识                 = 1
    使用自动存储器                         = Yes
    已启用自动调整大小                     = Yes
   文件系统高速缓存                       = No
   表空间状态                             = 0x'00000000'
      ....................................略.............................. .
```

- db2pd –tablespaces 命令(此命令会在第 12 章讲解)
- db2look –d <dbname> –l 命令(此命令会在第 13 章讲解)

下面我们举几个关于文件缓存的例子：

例 3-1 假定数据库和所有相关表空间容器位于 AIX JFS 文件系统上，并且发出了以下语句：

```
DB2 CREATE TABLESPACE DATA_SPACE MANAGED BY DATABASE USING (file '/db2data1' 800M)
```

在先前版本中，如果未指定该属性，那么数据库管理器将使用缓冲 I/O(FILE SYSTEM CACHING)作为 I/O 机制；对于版本 9.5，数据库管理器使用 NO FILE SYSTEM CACHING。

例 3-2 在以下语句中，NO FILE SYSTEM CACHING 子句指示对于此特定表空间，文件系统级高速缓存将 OFF。

```
CREATE TABLESPACE table space name ... NO FILE SYSTEM CACHING
```

例 3-3　以下语句对现有表空间禁用文件系统级高速缓存：

```
ALTER TABLESPACE table space name ... NO FILE SYSTEM CACHING
```

例 3-4　以下语句对现有表空间启用文件系统级高速缓存：

```
ALTER TABLESPACE table space name ... FILE SYSTEM CACHING
```

经过上面的讲解，建议创建表空间时，表空间的容器采用裸设备或支持并发 I/O 或直接 I/O 的文件系统。

3.2.6　OVERHEAD 和 TRANSFERRATE 设置

这两个参数用于确定查询优化期间的 I/O 成本。这两个值的测量单位都是毫秒，而且它们应当分别是所有容器开销和传送速率的平均值。开销是与 I/O 控制器活动、磁盘寻道时间和旋转延迟时间相关联的时间。传送速率是将一个页读入内存所必需的时间量。它们的默认值分别是 24.1 和 0.9。可以根据硬件规格计算这些值。

```
Transrate=(1/传送速率)*1000/1024000*4096(假设用 4KB 页大小)
Overhead=平均寻道时间+(((1/磁盘转速)*60*1000)/2)
```

而平均寻道时间、磁盘旋转速度和传送速率是由硬盘本身决定的(可以使用操作系统命令或向硬盘厂商获得底层硬盘物理特性)。

所以我们必须合理地设置这两个值以便让优化器了解我们底层存储的物理特性来制订最优的执行计划。

3.2.7　优化 RAID 设备上表空间性能

现在很多应用系统都把数据库存放在“独立磁盘冗余阵列”(RAID)设备上，要优化存放在 RAID 设备上的表空间性能，请遵循下列准则：

- 在一组 RAID 设备上创建表空间时，应该把表空间容器创建在多个 RAID GROUP 上。
- 考虑以下示例：您将 15 个 146GB 磁盘配置为 3 个 RAID-5 阵列，每个阵列包含 5 个磁盘。格式化以后，每个磁盘可容纳大约 136GB 数据。因此，每个阵列可存储大约 544GB(4 个活动磁盘×136GB)数据。如果表空间需要 300GB 存储空间，那么创建 3 个容器并将每个容器分别放在 3 个不同的 RAID 设备上。每个容器使用设备上的 100GB(300GB/3)，并且每个设备上保留 444GB(544GB－100GB)以提供附加表空间。

- 为表空间选择适当的扩展数据块(extent)大小。理想状态下，扩展数据块大小应该是磁盘底层 strip 大小的倍数，其中 strip 大小表示磁盘控制器向一个物理磁盘写入多少数据才转向下一个物理磁盘。选择应该是 strip 大小倍数的扩展数据块大小，以确保基于扩展数据块的操作(如预取时的并行顺序读取)不会争用相同的物理磁盘。
- 在上面我们举的那个示例中，如果 strip 大小为 64KB，而页大小为 16KB，那么适当的扩展数据块大小可能是 256KB。
- 使用 DB2_PARALLEL_IO 注册表变量来对所有表空间启用并行 I/O，并对每个容器指定物理磁盘数。在设置个变量之前，我们先了解这个变量的含义。

DB2_PARALLEL_IO 注册表变量用来确定每个容器的底层物理硬盘数以及对表空间上的并行 I/O 的影响。当我们为一个表空间设置多个容器或者设置 DB2_PARALLEL_IO 注册表变量时都会为表空间启动并行预取。并行预取请求的个数是由表空间容器个数和 DB2_PARALLEL_IO 同时决定的，每个预取请求都只能读取一个 EXTENT。如果未设置 DB2_PARALLEL_IO，那么表空间的并行度与容器数目相等。否则，表空间的并行度由表空间预取大小和 EXTENT 大小决定，我们建议大家将预取大小设置为 AUTOMATIC，这样就由 DB2 自动计算预取大小，让 DB2 选择最合适的并行度。

DB2_PARALLEL_IO=TablespaceID:[*n*]，如果没有指定 *n*，那么使用默认值 6(就是假定一个容器跨越 6 个 RAID 底层物理磁盘)。以下是 DB2_PARALLEL_IO 注册表变量如何影响预取大小的若干示例(假设已使用 AUTOMATIC 预取大小定义以下所有表空间)。

- DB2_PARALLEL_IO=*

“*”表示所有表空间均会启用并行 I/O，由于没有指定 n，所有表空间将使用每个容器磁盘数目等于 6 时的默认值。预取请求分解成“6×容器个数”个并行请求，每个请求的读取大小为 extent 大小。

- DB2_PARALLEL_IO=*:3

“*”表示所有表空间均会启用并行 I/O。“*:3”表示所有表空间将 3 作为每个容器的默认磁盘数目。预取请求分解成“3×容器个数”个并行请求，每个请求的读取大小为 extent 大小。

- DB2_PARALLEL_IO=*:3，1:1

“*”表示所有表空间均会启用并行 I/O。“*:3”表示所有表空间将 3 作为每个容器的磁盘个数，预取请求分解成“3×容器个数”个并行请求，每个请求的读取大小为 extent 大小。对于 ID 为 1 的表空间，预取请求分解成“1×容器个数”个并行请求，每个请求的读取大小为 extent 大小。

对于此示例中的情况，请将 DB2_PARALLEL_IO 设置为*:4，如图 3-17 所示。

图 3-17　DB2_PARALLEL_IO 设置为*:4

如果将表空间的预取大小设置为 AUTOMATIC，那么数据库管理器将使用您对 DB2_PARALLEL_IO 指定的物理磁盘数值来确定预取大小值。如果预取大小未设置为 AUTOMATIC，那么您可以手动设置此值，请考虑 RAID 条带大小，它是 strip 大小乘以活动磁盘数产生的值。考虑满足下列条目的预取大小值：

- 它等于 RAID 条带大小乘以 RAID 并行设备数(或此乘积的整数表示)。
- 它是扩展数据块大小的整数表示。

在上面示例中，可将预取大小设置为 768KB。此值等于 RAID 条带大小(256KB)乘以 表空间的 RAID 并行容器设备数(3)。它也是扩展数据块大小(256KB)的倍数。选择此预取大小意味着单个预取会涉及所有阵列中的所有磁盘。如果因为工作负载主要涉及大量扫描而希望预取程序更积极地工作，那么可改为使用此值的倍数，如 1536 KB(768KB×2)。

不要设置 DB2_USE_PAGE_CONTAINER_TAG 注册表变量。如之前所述，应使用等于 RAID 条带大小或其倍数的扩展数据块大小来创建表空间。但是，将 DB2_USE_PAGE_CONTAINER_TAG 设置为 ON 时，将使用单页容器标记，并且扩展数据块不会与 RAID 条带对齐。因此，在 I/O 请求期间可能需要访问比最优情况更多的物理磁盘。

3.2.8　合理设置系统临时表空间

系统临时表空间主要用于分组、排序、连接、重组和创建索引等。要确保系统临时表空间的最大页大小对于查询或定位更新来说足够大。

DB2 V9 中大记录标识符(RID)的使用增加了来自查询或定位更新的结果集的行大小。如果结果集中的行大小接近于现有系统临时表空间的最大行长度限制，那么可能需要创建

具有更大页大小的系统临时表空间。下面我们举个例子：

假如表 T1 具有 20 个字段，T2 具有 18 个字段，每行最大长度分别为 3500 字节和 3000 字节，它们能正常地存放在 4KB 表空间中。但是如果我们发出如下的这条 SQL 语句：

```
select T1.*,T2.* from T1,T2 where T1.id=T2.id
```

对于上面这条 SQL 语句，在临时表空间中一行的长度已经达到 6500(3500+3000)字节大小，这时原来的 4KB 的临时表空间已经不能存放，必须使用更大页大小的临时表空间。

所以要确保系统临时表空间的最大页大小对于查询或定位更新足够大，否则会显著影响性能。可以使用如下方法：

- 确定来自查询或定位更新的结果集的最大行大小。使用曾用来创建表的 DDL 语句来监控查询或者计算最大行大小。
- 检查结果集中的最大行大小是否适合系统临时表空间的页大小：

```
maximum_row_size>maximum_row_length-8 字节(单分区结构开销)
```

其中 maximum_row_size 是结果集的最大行大小，maximum_row_length 是基于所有系统临时表空间的最大页大小所允许的最大长度，应根据表空间页大小确定最大行长度。

- 创建一个系统临时表空间，其大小应至少比创建了表的表空间页大小大一个页大小(如果还没有这样大小的系统临时表)。例如，在 Windows 操作系统上，如果在一个具有 4KB 页大小的表空间中创建了表，那么使用 8KB 页大小创建额外系统临时表空间以备需要的时候使用：

```
CREATE SYSTEM TEMPORARY TABLESPACE tmp_tbsp PAGESIZE 8K
            MANAGED BY SYSTEM  USING ('d:\tmp_tbsp','e:\tmp_tbsp')
```

如果表空间页大小是 32KB，那么可以减少在查询中选择的信息或者分开这些查询以适合系统临时表空间页。例如，如果选择了表的所有列，那么可以改为仅选择真正需要的列或者选择某些列的一个子串来避免超出页大小限制。

3.3 缓冲池

缓冲池指的是从磁盘读取表和索引数据时，数据库管理器分配的用于高速缓存这些表或索引数据的内存区域。每个 DB2 数据库都必须具有至少一个缓冲池。数据库中的数据访问都需要经过缓冲池：读的数据需要先读到缓冲池才能提交给应用，写的数据也是要先写到缓冲池才能进行 I/O。缓冲池是影响数据库性能最大的参数，所以必须合理地设计缓

冲池。

创建数据库时，DB2 会自动地创建一个名为 IBMDEFAULTBP 的默认缓冲池，所有的表空间都共享该缓冲池。可以使用 CREATE BUFFERPOOL、DROP BUFFERPOOL 和 ALTER BUFFERPOOL 语句来创建、删除和修改缓冲池。SYSCAT.BUFFERPOOLS 目录视图记录数据库中所定义的缓冲池的信息。缓冲池的默认大小是 BUFFPAGE 数据库配置参数所指定的大小，但是可以通过在 CREATE BUFFERPOOL 命令中指定 SIZE 关键字来覆盖该默认值。足够的缓冲池大小是数据库拥有良好性能的关键所在，因为它可以减少磁盘 I/O 这一最耗时的操作。大型缓冲池还会对查询优化产生影响，因为更多的工作可在内存中完成，而无须进行 I/O。

3.3.1　缓冲池的使用方法

首次访问表中的数据行时，数据库管理器会将包含该数据的页放入缓冲池中。这些页将一直保留在缓冲池中，直到关闭数据库或者其他页需要使用某一页所占用的空间为止。缓冲池中的页可能正在使用，也可能没有使用，它们可能是脏页，也可能是干净页。

- 正在使用的页就是当前正在读取或更新的页。为了保持数据一致性，数据库管理器只允许一次只有一个代理程序更新缓冲池中的给定页。如果正在更新某页，那么它只能允许一个代理程序互斥地访问。如果正在读取该页，那么多个代理程序可以同时读取该页。
- “脏”页包含已更改但尚未写入磁盘的数据。
- 将一个已更改的页写入磁盘之后，它就是一个“干净”页，并且可能仍然保留在缓冲池中。

大多数情况下，调整数据库涉及到设置用于控制将数据移入缓冲池以及等待将数据从缓冲池写入磁盘的配置参数。如果最近的代理程序不需要页空间，那么可以将页空间用于新应用程序中的新页请求。额外的磁盘 I/O 会使数据库管理器性能下降。

可使用数据库监控工具来计算缓冲池命中率，缓冲池命中率可帮助您调整缓冲池。这部分内容我们会在第 9 章讲解。

3.3.2　缓冲池和表空间之间关系

设计缓冲池时，需要了解表空间与缓冲池之间的关系。每个表空间都与一个特定的缓冲池相关。IBMDEFAULTBP 是默认缓冲池。数据库管理器还会分配下列系统缓冲池：IBMSYSTEMBP4K、IBMSYSTEMBP8K、IBMSYSTEMBP16K 和 IBMSYSTEMBP32K(以前称为“隐藏缓冲池”)。要使另一个缓冲池与表空间相关，那么该缓冲池必须存在并且它们具有相同的页大小。关联是在使用 CREATE TABLESPACE 语句创建表空间时定义的，

但以后可使用 ALTER TABLESPACE 语句更改此关联。

如果拥有多个缓冲池，那么可以配置数据库使用更多的内存，以改善整体性能。例如，对于 OLAP 类型的应用，我们建议采用一个大的缓冲池，以利于大块顺序读取；用于联机事务应用程序的表空间可以根据业务特点使用多个小的缓冲池，以便可以更长时间地高速缓存应用程序所使用的数据页，便响应时间更快。

图 3-18 是一个表空间和缓冲池设计的例子。

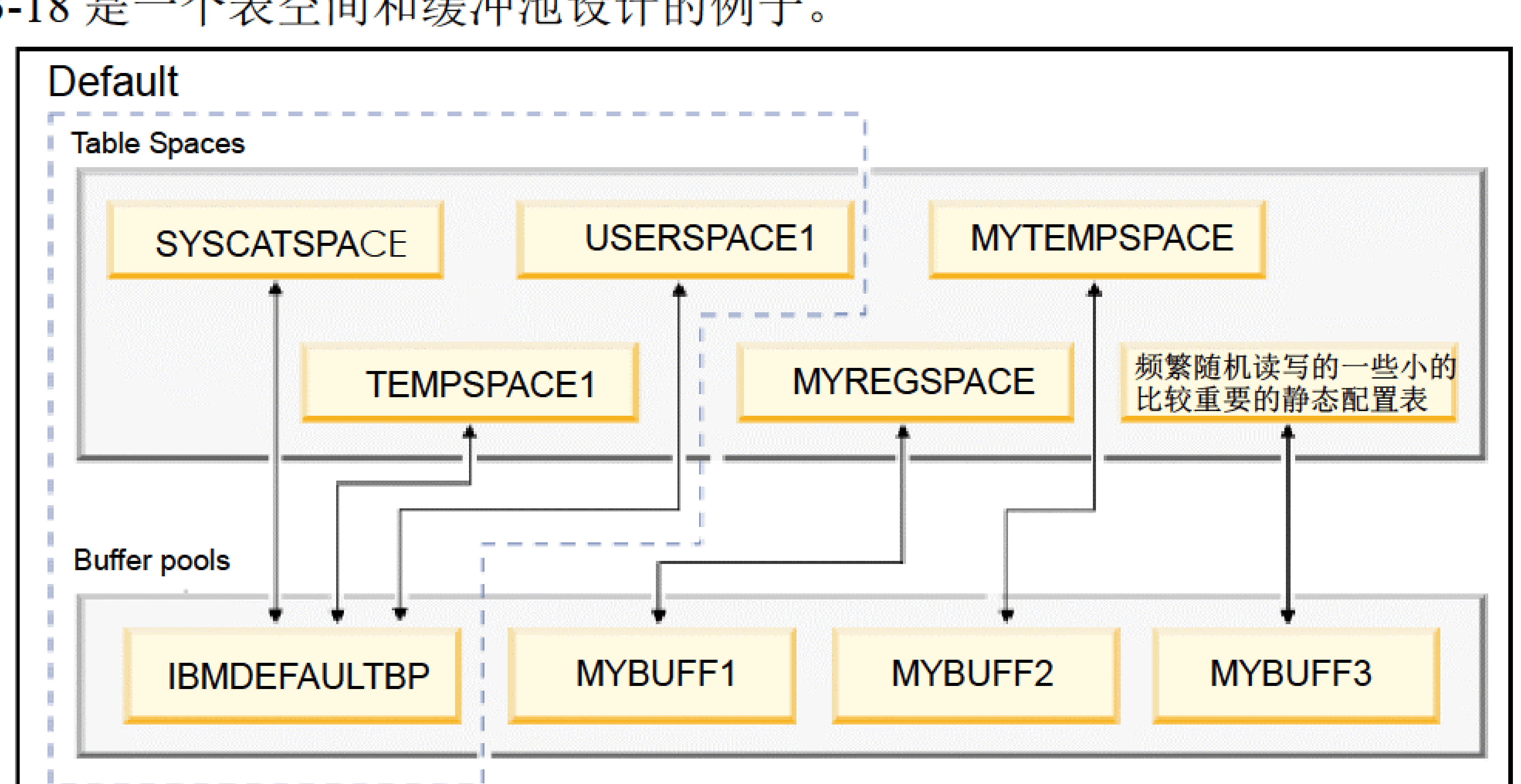


图 3-18　表空间和缓冲池关系设计示例

该数据库有 6 个表空间：数据库创建时默认生成的 3 个表空间：系统编目表空间、系统临时表空间和 USERSPACE1(使用默认的缓冲池 IBMDEFAULTBP)；用户定义的常规表空间 MYREGSPACE(使用 MYBUFF1 缓冲池)、MYTEMPSPACE(使用 MYBUFF2 缓冲池)，最后一个表空间用于放置一些频繁随机读写的比较重要的静态配置表(使用 MYBUFF3 缓冲池)。在图 3-18 中，我们没有看到为 LONG 表空间设置的缓冲池，这是因为大对象的读取不能经过内存，从磁盘直接读取。

3.3.3　缓冲池维护

缓冲池页大小

默认缓冲池的页大小是在使用 CREATE DATABASE 命令时设置的。此默认值表示所有将来 CREATE BUFFERPOOL 和 CREATE TABLESPACE 语句的默认页大小。如果在创建数据库时不指定页大小，那么默认页大小是 4KB。

注意：

如果确定数据库需要 8KB、16 KB 或 32 KB 的页大小，那么必须至少定义一个具有相

匹配的页大小并且与数据库中的表空间相关联的缓冲池。选择用于缓冲池的页大小是很重要的，这是因为创建缓冲池之后就不能改变页大小了。

基于块(block)的缓冲池

DB2 允许您留出缓冲池的一部分(最高可达 98%)用于基于块的预取操作。基于块的 I/O 可以通过将块读入相邻的内存区而不是将它分散装入单独的页，来提高预取操作的效率。每个缓冲池的块大小必须相同，并且由 BLOCKSIZE 参数进行控制。该值等于块的大小(单位为页)，取值范围从 2 到 256，默认值为 32。

注意：
基于块的缓冲池主要用于数据仓库、DSS 之类的连续大块读写的应用中。

在创建新的缓冲池之前，应解决下列问题：

- 想要使用什么缓冲池名称？
- 是立即创建缓冲池，还是在下一次取消激活然后重新激活数据库之后创建缓冲池？
- 希望缓冲池的页大小是多大？
- 是将缓冲池设为固定大小，还是由数据库管理器自动调整缓冲池大小以对工作负载作出响应？建议您在创建缓冲池期间不指定 SIZE 参数，从而允许数据库管理器自动调整缓冲池。
- 您是否想保留一部分缓冲池用于基于块的 I/O？
- 设计缓冲池时，还应根据机器上已安装的内存量以及与数据库管理器在同一机器上同时运行的其他应用程序所需要的内存来考虑内存要求。当没有足够内存来保存所访问的所有数据时，操作系统就会进行数据交换。将某些数据写入或交换到临时磁盘存储器中以为其他数据腾出空间时就会进行数据交换。当需要临时磁盘存储器上的数据时，又会将数据交换回内存中。

创建缓冲池

恰当地定义缓冲池是拥有一个运行良好的系统的关键之一。对于 32 位操作系统，知道内存的寻址空间十分重要(AIX 是 1.75 GB；Linux 是 1.75 GB；Sun Solaris 是 3.35 GB；HP-UX 是大约 800MB；Windows 是 2-3GB)。64 位系统没有这样的界限。

使用 CREATE BUFFERPOOL 语句来定义数据库管理器要使用的新缓冲池。以下是基本 CREATE BUFFERPOOL 语句的一个示例：

```
CREATE BUFFERPOOL BP3 SIZE 2000 PAGESIZE 8K
```

创建缓冲池的两个关键参数是 IMMEDIATE 和 DEFERRED。当使用 IMMEDIATE 参数时，将立即更改缓冲池大小，而不必等到下一次激活数据库时才生效。默认情况下，新的缓冲池是使用 IMMEDIATE 关键字创建的。对于立即请求，不需要重新启动数据库，将立即激活缓冲池。如果数据库共享内存不足以分配新空间，那么会延迟(DEFERRED)运行该语句。

如果您发出 CREATE BUFFERPOOL DEFERRED，那么不会立即激活缓冲池；将在下一次启动数据库时创建缓冲池。在重新启动数据库之前，任何新的表空间都将使用现有缓冲池，即使创建该表空间时显式使用延迟缓冲池也是如此。

创建缓冲池时，应查看机器上是否有足够的内存用于已创建的所有缓冲池。要综合考虑操作系统上别的应用和操作系统本身的内存需求。

修改缓冲池

有许多理由要修改缓冲池，例如，为了启用自调整内存功能。为此，可以使用 ALTER BUFFERPOOL 语句。可以修改缓冲池的如下属性：

- 启用缓冲池自调整功能，从而允许数据库管理器根据工作负载调整缓冲池大小；
- 修改基于块的 I/O 的缓冲池的块区域；
- 修改部分缓冲池的大小。

使用 ALTER BUFFERPOOL 语句来改变缓冲池对象的单个属性。例如：

```
ALTER BUFFERPOOL buffer pool name SIZE number of pages
```

buffer pool name 是缓冲池名称，number of pages 是要分配给此特定缓冲池的新页数。也可以使用值 – 1，它指示缓冲池大小应该是在 buffpage 数据库配置参数中设置的值。

查看缓冲池属性

通过查询 SYSCAT.BUFFERPOOLS 系统视图可以列出缓冲池信息：

```
SELECT * FROM SYSCAT.BUFFERPOOLS
BPNAME          BUFFERPOOLID NGNAME             NPAGES      PAGESIZE    ES
--------------- ------------ ------------------ ----------- ----------- --
IBMDEFAULTBP               1 -                          250        4096 N
  1 record(s) selected.
```

要找出哪个缓冲池被分配给了表空间，请运行下面这个查询：

```
SELECT TBSPACE, BUFFERPOOLID FROM SYSCAT.TABLESPACES
TBSPACE          BUFFERPOOLID
```

```
------------------ ------------
SYSCATSPACE                  1
TEMPSPACE1                   1
USERSPACE1                   1
  3 record(s) selected.
```

可以在上一个查询中找到 BUFFERPOOLID，该查询使您能够看到每个表空间与哪个缓冲池相关联。

删除缓冲池

删除缓冲池时，应确保没有任何表空间已指定给这些缓冲池。不能删除 IBMDEFAULTBP 缓冲池。

可以使用 DROP BUFFERPOOL 语句来删除缓冲池，如下所示：

```
DROP BUFFERPOOL <buffer pool name>
```

3.3.4 缓冲池设计原则

缓冲池的命中率

使用多个用户表空间的最重要原因是管理缓冲池的命中率。一个表空间只能与一个缓冲池相关联，而一个缓冲池则可用于多个表空间。

缓冲池调优的目标是帮助 DB2 尽可能好地利用可用于缓冲池的内存。整个缓冲池大小对 DB2 性能有巨大影响，这是因为缓存大量的页可以显著地减少 I/O 这一最耗时的操作。但是，如果总的缓冲池设置太大，并且没有足够的物理内存来分配给它们，那么系统将会使用每种页大小最少的缓冲池，性能就会急剧下降。要计算最大的缓冲池大小，需要综合考虑 DB2、操作系统以及其他任何应用程序内存的使用率。一旦确定了 DB2 总的可用内存大小，就可以将这个区域划分成不同的缓冲池以提高命中率。如果有一些具有不同页大小的表空间，那么每种页大小必须至少有一个缓冲池。

拥有多个缓冲池可以最大限度地将数据保存在缓冲池中。例如，让我们假设一个数据库有许多频繁使用的小型表，这些表通常全部都位于缓冲池中，因此访问起来就非常快。现在让我们假设有一个针对非常大的表运行的查询，它使用同一个缓冲池并且需要读取比总的缓存池大小还多的页。当查询运行时，之前来自这些频繁使用的小型表的页将会丢失，这使得再次需要这些数据时就必须重新读取它们。

如果小型表拥有自己的缓冲池，那么它们就必须拥有自己的表空间，在这种情况下其他的查询就不能覆盖它们的页。这有可能产生更好的整体系统性能，虽然这会对大型查询造成一些小的负面影响。经常性地进行调优是为了实现整体的性能提高，而且时常需要在不同的系统功能之间作出权衡。区分功能的优先级并记住总吞吐量和使用情况，同时对系统性能进行调整，这是非常重要的。

DB2 V8 所引入的新功能能够在不关闭数据库的情况下更改缓冲池大小。带有 IMMEDIATE 选项的 ALTER BUFFERPOOL 语句会立刻生效，只要数据库共享的内存中有足够的保留空间可以分配给新空间。可以使用这个功能，根据使用过程中的周期变化(例如从白天的交互式使用转换到夜间的批处理工作)来调优数据库性能。

关于如何监控和调整缓冲池，请详细参考本书“第 9 章：DB2 数据库监控”。

确定有多少缓冲池

对于数据库中一个表空间所使用的每一种页面大小，都需要至少一个缓冲池。通常，默认的 IBMDEFAULTBP 缓冲池是留给系统编目的。为处理表空间的不同页面大小和行为，需创建新的缓冲池。

建议为每种页面大小使用一个缓冲池，对于 OLAP/DSS 类型的工作负载更是如此。DB2 在其缓冲池的自我调优方面十分擅长，并且会将经常被访问的行放入内存，因此多数情况下对于每一种页的大小创建一个缓冲池就足够了(这一选择也避免了管理多个缓冲池的复杂性)。

如果时间允许，并且需要进行改进，那么您可能希望使用多个缓冲池。其思想是将访问最频繁的行放入一个缓冲池中。在那些随机访问或者很少访问的表之间共享一个缓冲池可能会给缓冲池带来“污染”，因为有时候要为一个本来可能不会再去访问的行消耗空间，甚至可能将经常访问的行挤出到磁盘上。如果将索引保留在它们自己的缓冲池中，那么在索引使用频繁的时候(例如，索引扫描)还可以显著地提高性能。

这与我们对表空间的讨论是紧密联系的，因为要根据表空间中表的行为来分配缓冲池。如果采用多缓冲池的方法，对于初学者来说使用 4 个缓冲池比较合适：

- 一个中等大小的缓冲池，用于临时表空间；
- 一个大型的缓冲池，用于索引表空间；
- 一个大型的缓冲池，用于那些包含经常要访问的表的表空间；
- 一个小型的缓冲池，用于那些包含访问不多的表、随机访问的表或顺序访问的表的表空间。

对于 DMS 只包含 LOB 数据的表空间，可以为其分配任何缓冲池，因为 LOB 不占用缓冲池空间。

确定为缓冲池分配的内存

千万不要为缓冲池分配多于所能提供的内存，否则就会招致代价不菲的操作系统内存分页(memory paging)。通常来讲，如果没有进行监控，要想知道一开始为每个缓冲池分配多少内存是十分困难的。

对于 OLTP 类型的工作负载，一开始将 25%(仅为参考，实际大小请参考自己操作系统上的内存资源和运行在该操作系统上的应用情况)的可用内存分配给缓冲池比较合适。

对于 OLAP/DSS，经验法则告诉我们，应该将 40%(仅为参考，实际大小请参考自己操作系统上的内存资源和运行在该操作系统上的应用情况)的可用内存分配给一个缓冲池(假设只有一种页面大小)，同时监控排序情况，并对 SORTHEAP 作相应调整。

使用基于块(block-based)的缓冲池

对于有连续读写频繁的 OLAP 查询可以得益于基于块的缓冲池。默认情况下，所有缓冲池都是基于页的，这意味着预取操作将把磁盘上相邻的页放入到不相邻的内存中。而如果采用基于块的缓冲池，则 DB2 将使用块 I/O 一次将多个页读入缓冲池中，这样可以显著提高顺序预取的性能。

一个基于块的缓冲池由数据页和一个扩展数据块同时组成。CREATE 和 ALTER BUFFERPOOLSQL 语句的 NUMBLOCKPAGES 参数用于定义块内存的大小，而 BLOCKSIZE 参数则指定每个块的大小，即在一次块 I/O 中从一个磁盘读取的页的数量。

共享相同扩展数据块大小的表空间应该成为一个特定的基于块的缓冲池的专门用户。将 BLOCKSIZE 设置为等于正在使用该缓冲池的表空间的 EXTENTSIZE 的整数倍。下面我们举一个创建基于块的缓冲池的例子：

```
C:\>db2 create bufferpool block_bp  size 40960 numblockpages 20480 blocksize 128
DB20000I  SQL 命令成功完成。
```

确定分配多少内存给缓冲池内的块区要更为复杂一些。如果碰到大量的顺序预取操作，那么您很可能会想要更多基于块的缓冲池。NUMBLOCKPAGES 应该是 BLOCKSIZE 的倍数，并且不能大于缓冲池页面数量的 98%。建议开始先将它设小一点(不大于缓冲池总共大小的 15%或刚好 15%)。在后面还可以根据快照监视(snapshot monitor)对其进行调整。

3.4　本章小结

本章我们讲解了如何设计、规划和创建数据库、表空间和缓冲池。良好的设计是整个应用系统高性能运行的基石。希望我们了解数据库的特性，用最适合我们业务需求的技术来进行数据库的物理设计和逻辑设计。随着数据库技术的发展，数据库中有很多新技术克服了以往技术的一些缺点，所以我们必须了解这些新技术并合理地运用它。就像在 DB2 V9 中我们建议大家创建基于自动存储的表空间，并且在创建表空间时尽量创建大型(large)表空间。

第 4 章

访问数据库

当数据库创建后，我们应当如何访问数据库呢？我们可以通过 DB2 数据库本身提供的管理工具——GUI 图形化界面来访问数据库；也可以通过类似 DOS 的命令行窗口 DB2 CLP 来访问数据库；另外，还可以通过编写程序来访问数据库。需要注意的是，如果您想在远程客户端访问 DB2 数据库，那么就必须**配置 DB2 服务器通信和客户端通信**。

本章我们将对如何访问数据库进行探讨，主要讲解如下内容：

- DB2 GUI 图形化界面
- DB2 CLP 命令行窗口
- 配置 DB2 服务器通信
- 配置客户端通信

4.1 访问 DB2

图 4-1 给我们展示了访问 DB2 数据库的各种接口。我们可以通过 DB2 CLP 访问数据库；也可以通过 DB2 的管理工具——图形化界面访问数据库；还可以通过应用编程接口编写程序的方式访问数据库。如果远程客户端需要访问 DB2 数据库，就需要配置服务器通信和客户端通信。下面我们将分别讲解这些接口。

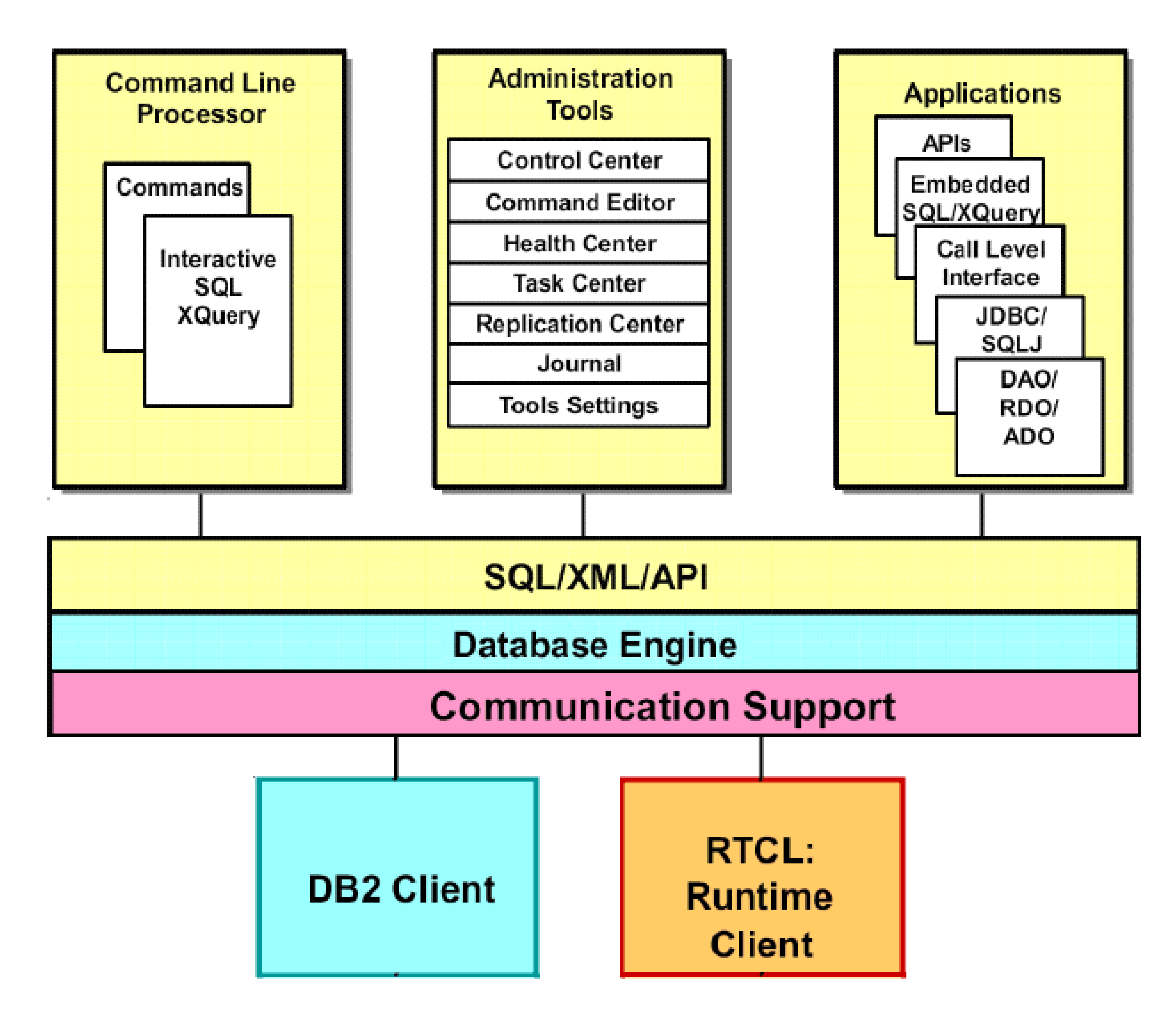


图 4-1　访问 DB2 数据库的接口

4.2　DB2 图形化操作环境

在进行数据库设计时，常常需要在 DB2 图形化环境下进行操作。DB2 图形化环境提供了大量的图形化工具，因此，在创建 DB2 数据库之前，就非常有必要了解并熟悉这一图形化操作环境。图 4-2 对 DB2 的图形化工具进行了总结。

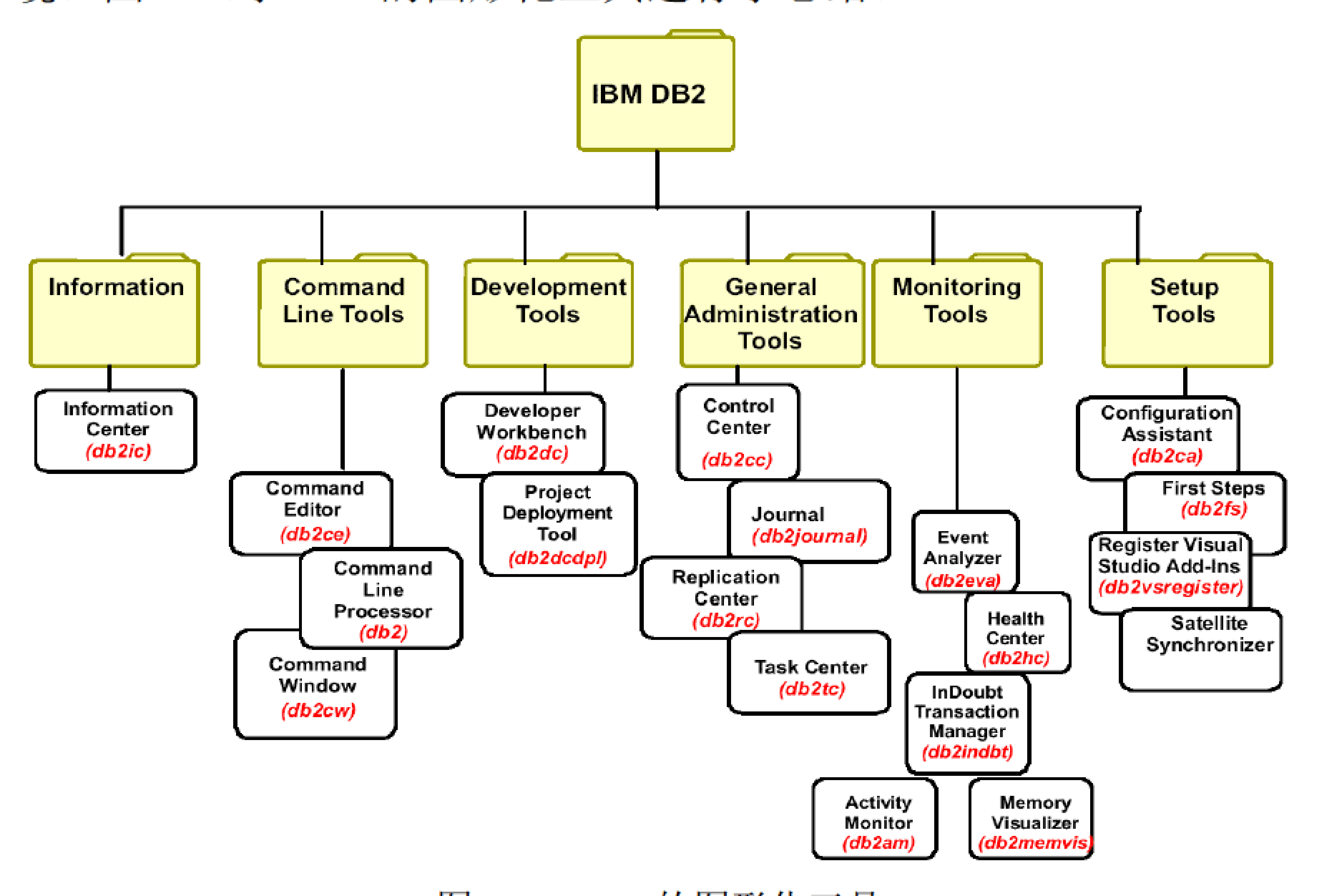


图 4-2　DB2 的图形化工具

图 4-3 是 Windows 上调用 DB2 图形化工具的菜单，下面我们将分别详细讲解每一种图形管理工具。

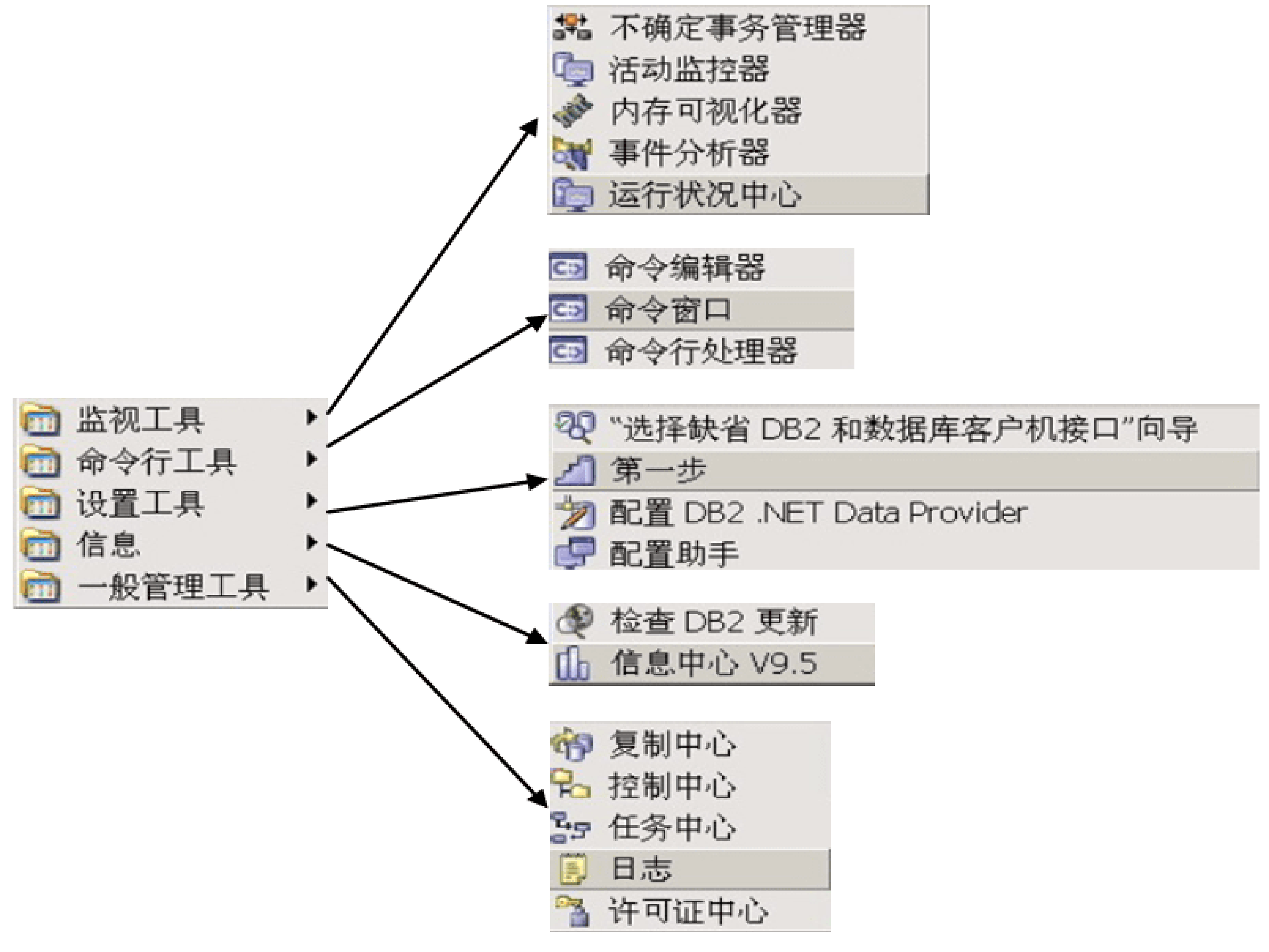


图 4-3　调用 DB2 图形化工具的菜单

控制中心

控制中心(Control Center)是 DB2 服务器的中心管理点，是 DB2 管理工具的核心，绝大多数管理任务和对其他管理工具的存取都可以通过控制中心来完成，如图 4-4 所示。

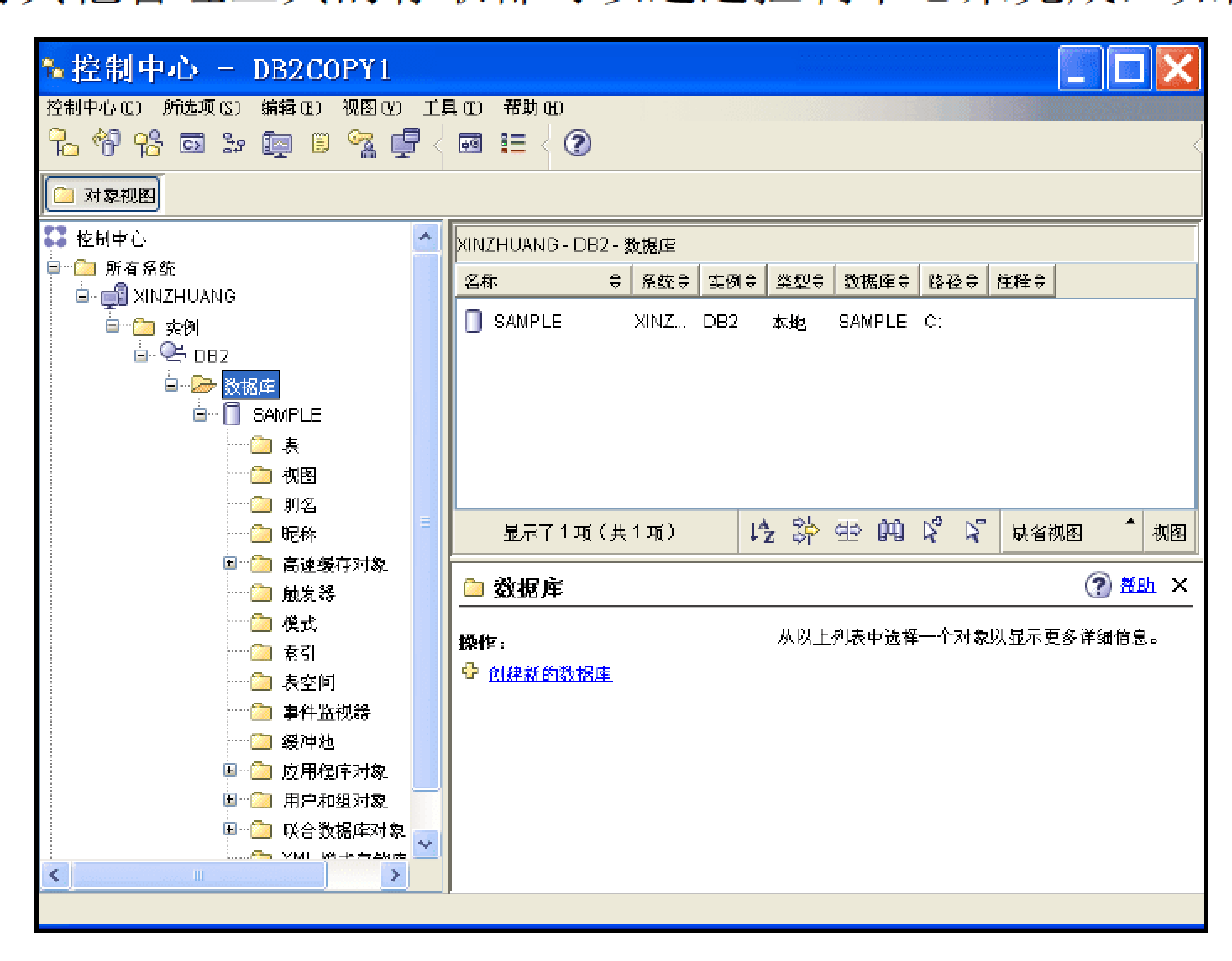


图 4-4　控制中心

在 Windows 平台上，可以依次从开始菜单选择“IBM DB2”，接着选择“一般管理工具”，最后选择“控制中心”来启动控制中心。也可以使用 db2cc 命令来启动控制中心。

控制中心依赖于数据库管理服务器(Database Administration Server，或简称为 DAS)。DAS 帮助控制中心调度作业在数据库服务器上的运行，并管理远程数据库服务器上的对象以及其他任务。

控制中心主要为数据库管理员提供下列功能：

- 添加 DB2 系统、数据库和数据库对象到对象树中；
- 管理数据库对象。包括创建、更改和删除数据库、表空间、表、视图、索引、触发器和模式。您还可以管理系统、实例、用户、组、别名、用户定义类型(UDT)、用户定义函数(UDF)、应用程序、程序包以及复制对象；
- 管理数据。您可以加载、导入或导出数据、重组数据并收集统计信息；
- 调度作业来自动运行；
- 备份和恢复数据库；
- 配置实例和数据库；
- 分析查询和可视化解释(Visual Explain))。使用可视化解释分析查询，查看访问计划；
- 监视并调优性能。您可以打开统计表(查看查询的执行路径)，启动事件和快照监视、生成数据库对象或命令的 SQL 或 DDL 并查看 DB2 对象之间的关系；
- 启动其他工具，如命令编辑器和健康中心；
- 更改整个控制中心用于显示菜单和文本的字体。

第一步

当完成 DB2 服务器端的软件安装之后，在默认配置下，系统会自动启动“第一步”，如图 4-5 所示。此时，DB2 服务器上并不存在任何数据库，但用户可以通过单击“第一步”窗口中的“创建样本数据库”链接来创建自己的第一个 DB2 数据库。



图 4-5 “第一步”窗口

配置助手

客户端如何能够识别远程数据库服务器呢？DB2 采取的办法是将远程数据库服务器端的节点信息、数据库信息分别写入客户端本地的 SQLNODIR 文件和 SQLDBDIR 文件，这样就可以将远程的数据库服务器映射到本地，这一过程称为编目(catalog)。而在编目之前，客户端必须能够与 DB2 服务器通信，客户机系统的数据库管理员必须建立该数据库管理系统与 DB2 服务器的通信，建立的方式则取决于操作系统和 DB2 服务器所用的通信协议。

DB2 配置助手(CA)可以用来对客户端应用程序所使用的数据库服务器进行配置和维护，它可以在客户端通过图形化的方式对远程数据库服务器端的节点和数据库进行编目，使用户免受命令行方式下对数据库进行手工编目之苦。如图 4-6 所示，配置助手(CA)可以用来维护客户端当前连接的 DB2 数据库配置参数，维护当前连接的 DB2 数据库管理服务器参数，配置新的数据库连接，绑定应用程序，以及导入和导出配置信息。

在 Windows 平台上，可以依次从开始菜单选择“IBM DB2”，接着选择“设置工具”，最后选择“配置助手”来启动配置助手。也可以使用 db2ca 命令来启动配置助手。

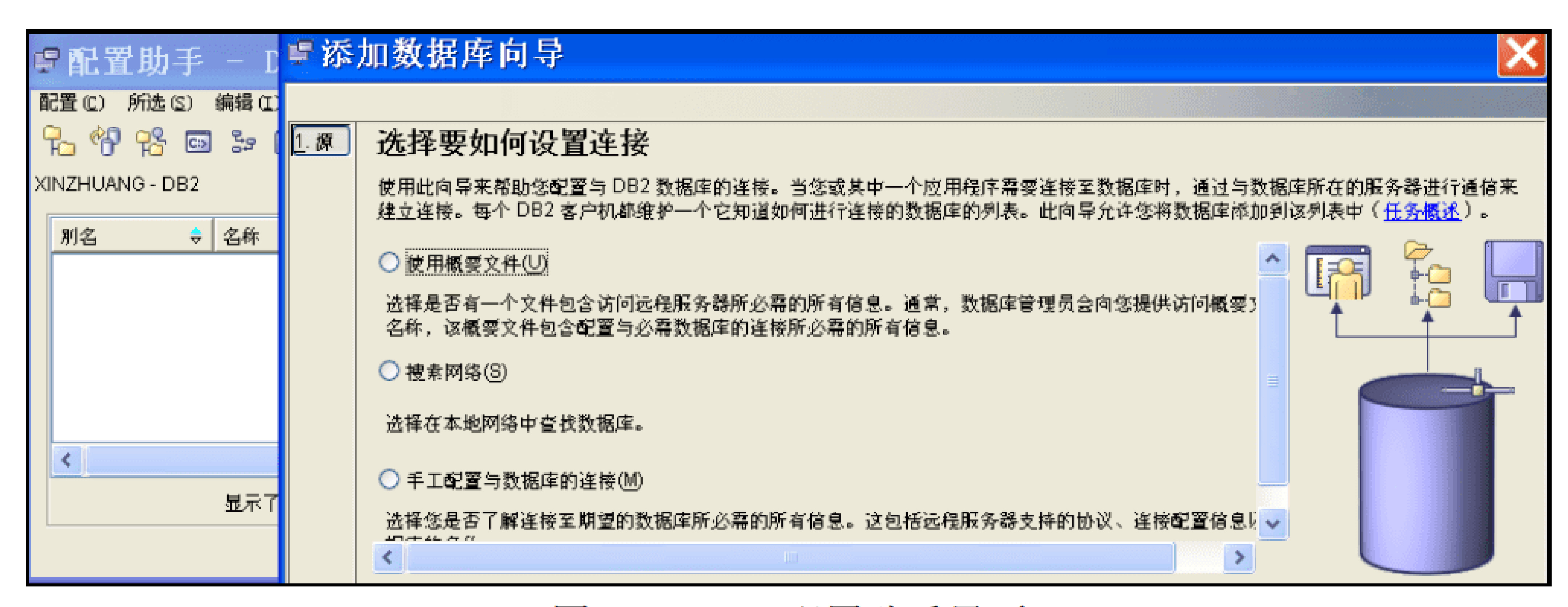


图 4-6　DB2 配置助手界面

复制中心

复制中心用于管理 DB2 数据服务器和其他关系数据库(DB2 或非 DB2)之间的复制。可以使用该工具创建复制定义和管理 Capture、Apply 和 Monitor 程序。我们可以在复制中心设置 SQL 复制、Q 复制等，如图 4-7 所示。

命令编辑器

命令编辑器是用来输入 DB2 命令的图形化工具，如图 4-8 所示。命令编辑器就是一个图形化的命令处理器(CLP)，可以在命令编辑器内输入 DB2 命令或调用现成的命令脚本，执行后可以查看输出结果。它相当于 DB2 CLP 命令行的图形化界面实现。

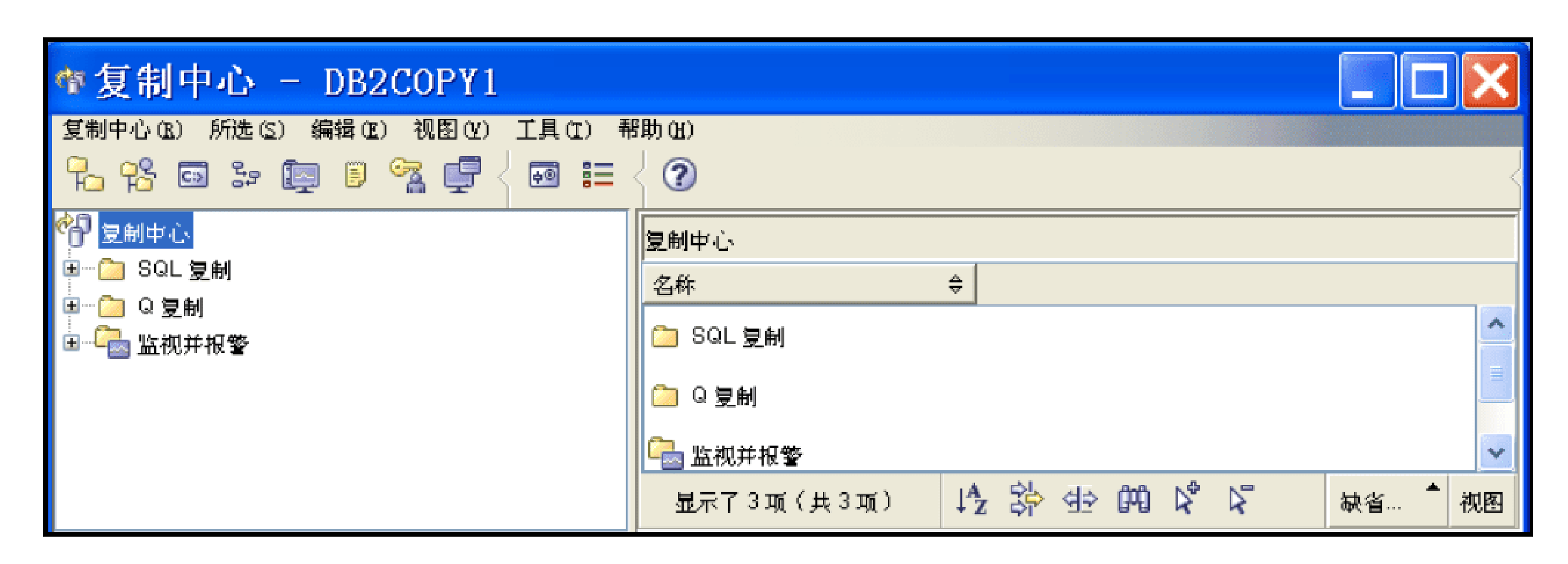


图 4-7 复制中心的功能

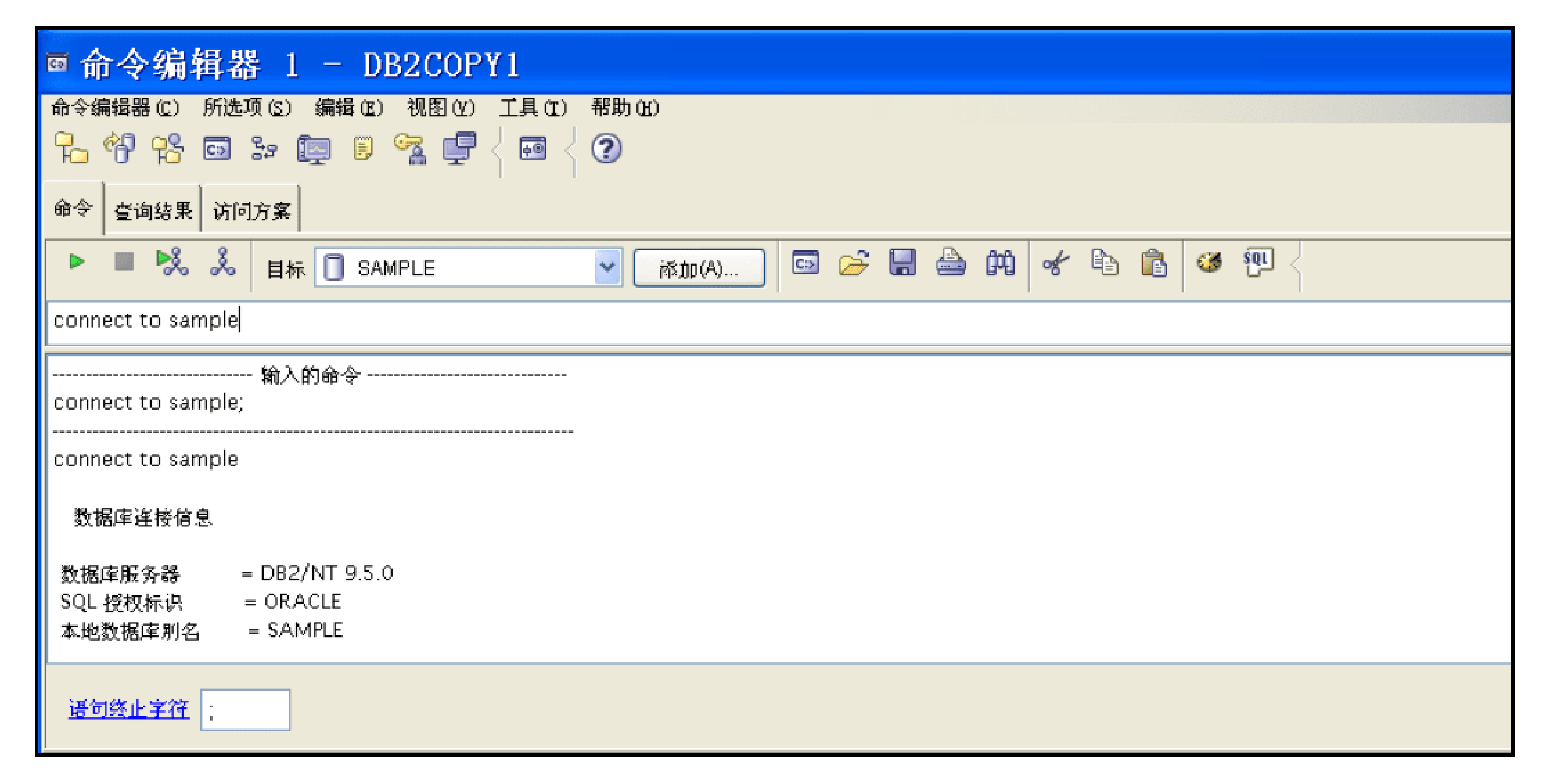


图 4-8 命令编辑器

命令编辑器可以将已输入的命令作为脚本保存到脚本中心中，也可以调用已保存在脚本中心内的脚本。另外，命令编辑器的一个特别有用的功能就是，用户可以利用它查看 SQL 语句的存取计划。存取计划中包含了 SQL 语句执行情况的统计结果，这样用户可以通过命令编辑器为 SQL 语句生成存取计划，并按照可视化的形式表现出来。

任务中心

任务中心可以用来安排、运行任务并通知人们已完成任务的状态。任务是一种附带相关的失败或成功条件、调度计划和通知的脚本，脚本中可以包含 DB2 命令、SQL 语句或操作系统命令。

任务中心中还可以创建组合任务以根据多个任务的结果来定义操作。组合任务与任务中心中的其他任务不同，因为没有任何命令脚本与组合任务直接关联。组合任务包含了已在任务中心中定义的任务。创建组合任务的优点是可创建依赖于多个任务结果的任务操作。任务中心如图 4-9 所示。

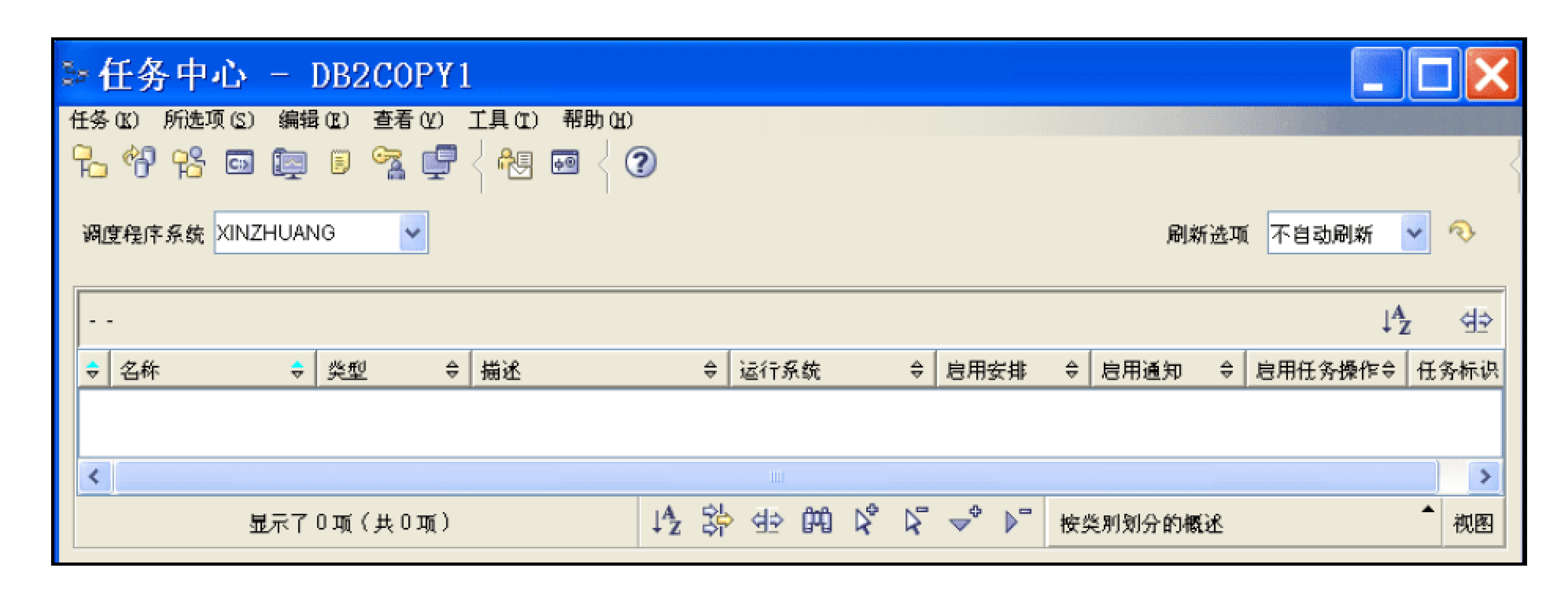


图 4-9　任务中心

日志

利用日志能够监视作业和查看结果，如图 4-10 所示。可以通过从控制中心工具栏选择日志图标来启动日志。从日志中还可以显示恢复历史和 DB2 警告消息。日志允许监视暂挂的作业、正在运行的作业和作业历史；查看结果；它还显示 DB2 消息记录。

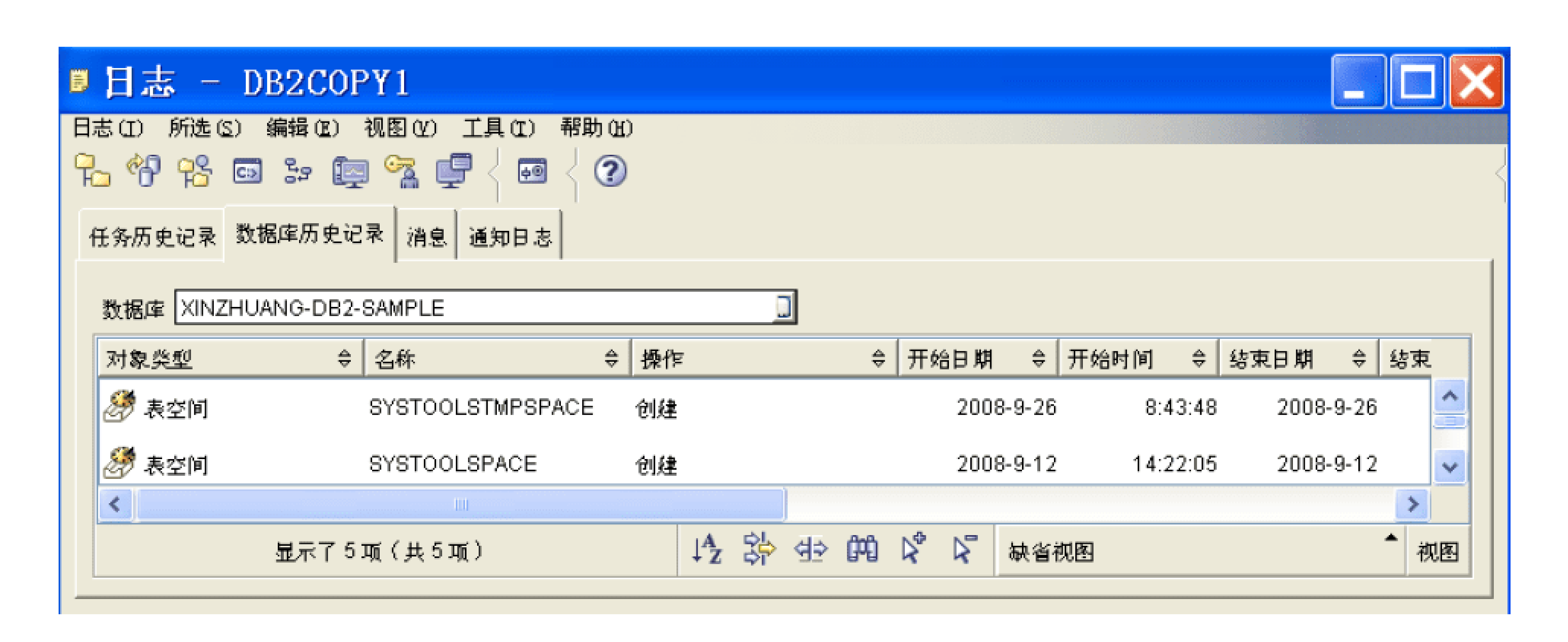


图 4-10　“日志”窗口

许可证中心

许可证中心显示 DB2 许可证状态和安装在系统上的 DB2 产品的使用情况，如图 4-11 所示。另外，它用来配置系统以便进行适当的许可证监视。可以用许可证中心来添加新的许可证，设置并发的用户策略，将先试后买的许可证升级为生产版许可证，用户也可

以浏览当前安装在 DB2 系统上的许可证信息，例如产品名称、产品的版本、过期时间，以及允许的用户数目等信息。此外，还可以通过在命令行使用 db2licm 命令来维护 DB2 许可证。



图 4-11 “许可证中心”窗口

信息中心

使用信息中心可以查找关于任务、书籍、参考资料、故障排除、样本程序以及相关 Web 站点的信息，如图 4-12 所示。



图 4-12 信息中心

这个中心提供了丰富的 DB2 信息。在 Windows 环境中您可以从控制中心或者从开始菜单启动信息中心。也可以使用 db2ic 命令快速启动信息中心。

内存可视化器

使用内存可视化器来监视 DB2 数据库的内存使用情况，如图 4-13 所示。

不确定事务管理器

使用不确定事务管理器来处理不确定的全局事务，如图 4-14 所示。例如，中断的通信会让事务做好准备，但还不会提交或回滚。它通常用于诊断分布式数据库的两阶段提交。

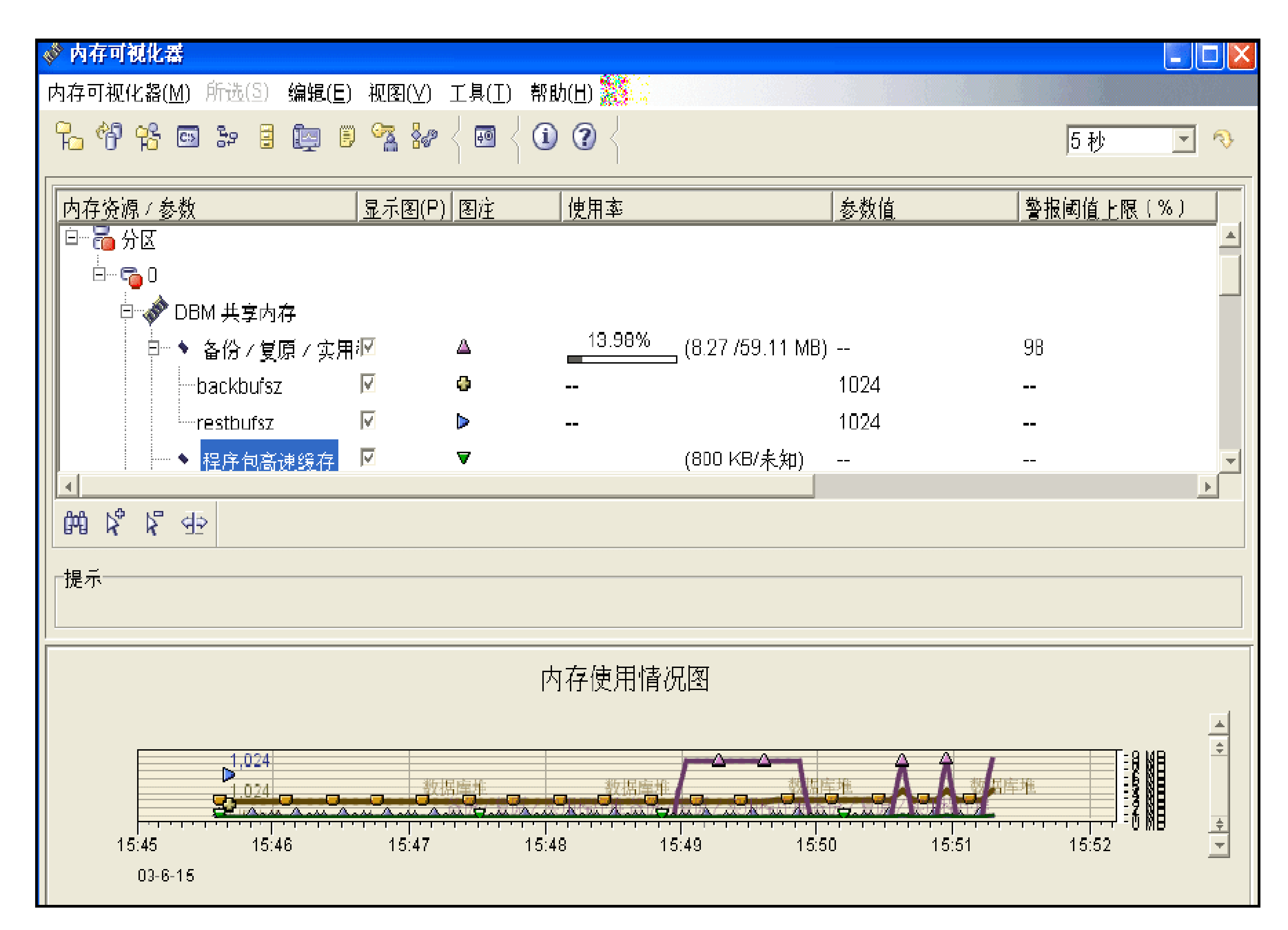


图 4-13　内存可视化器

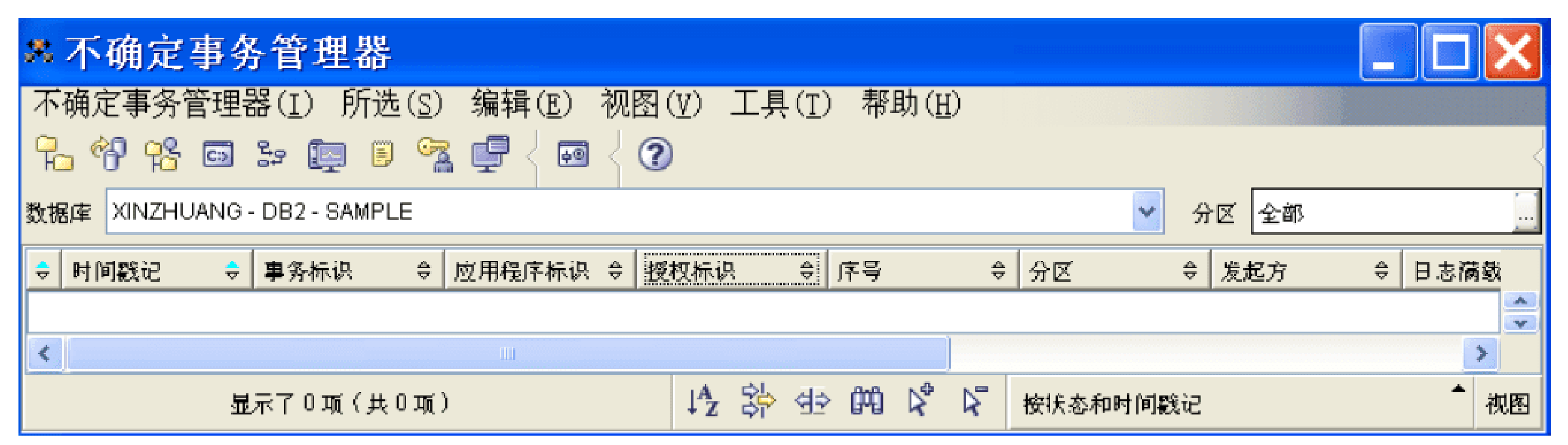


图 4-14　不确定事务管理器

活动监视器

DB2 UDB V8.2 中增加了一个新的图形化工具，称为 Activity Monitor(活动监视器)。Activity Monitor 可以用来监视应用程序性能、应用程序并发性、资源消耗和 SQL 语句的使用情况，如图 4-15 所示。它可以帮助用户诊断数据库性能问题(比如等待锁状态)，以及调优查询来优化对数据库资源的使用。Activity Monitor 还提供 DB2 自动生成的许多报告。

其实，活动监视器本质上是对底层调用的一些监控命令和函数的封装，只不过是以图形化的直观方式显示。在这里，我建议大家熟练掌握监控数据库性能的手工命令，这是因为在实际的数据库生产环境中大多数没有配置图形化环境；并且，真正的高手很少依赖图形化界面。尽管如此，活动监视器不失为一个好的监控维护工具。



图 4-15　Activity Monitor -运行时分析界面

运行状况中心

该工具可以显示客户机上编目的所有实例的数据库系统健康运行状况。“运行状况中心”收集编目实例的信息，并在所有中心或“健康中心”主视图中提供潜在或现有问题的文本或图形通知。可以使用界面来查看每条警告的详细信息，如图 4-16 所示。该工具对如何解决问题提出建议，并提供了界面以应用解决方案。“运行状况中心”使用简易，DB2 的初学者也能够使用它成功判断问题并进而获取解决方案。

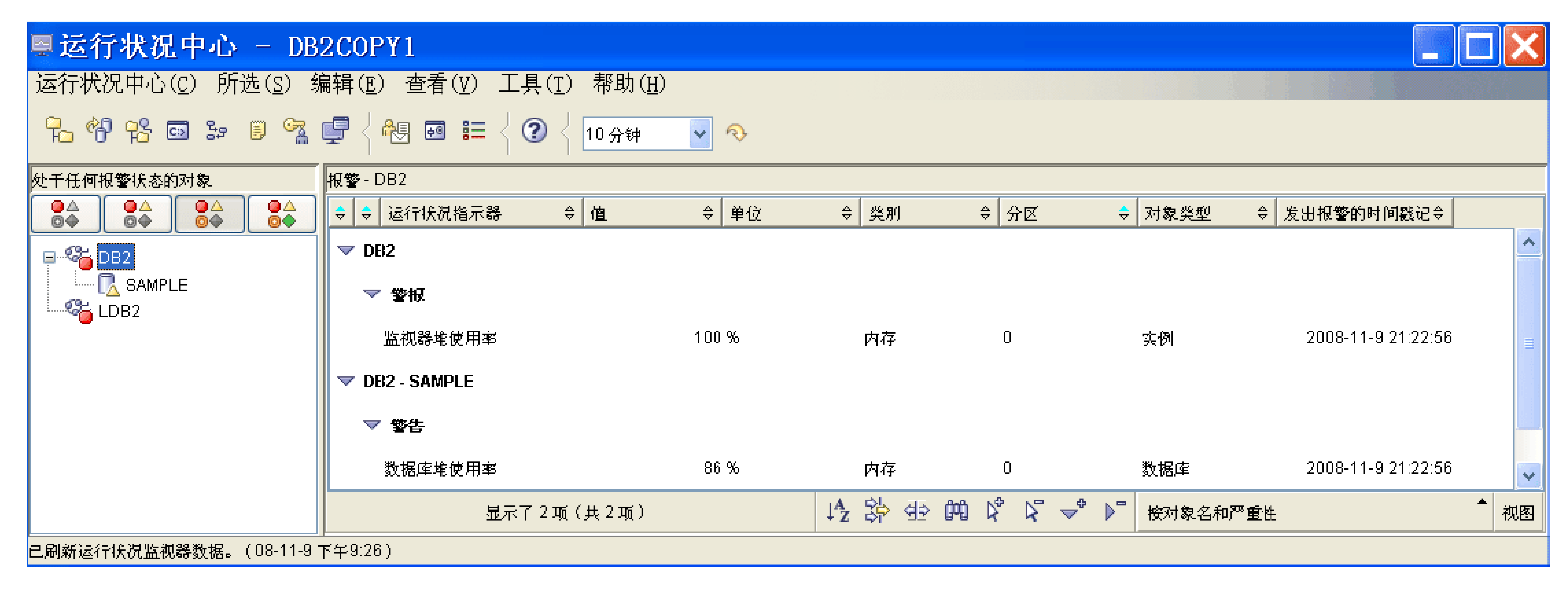


图 4-16　运行状况中心

事件分析器

这个管理工具只有在 Windows 平台才有，它可以用来对事件监视器的监控结果进行图形化分析，如图 4-17 所示。

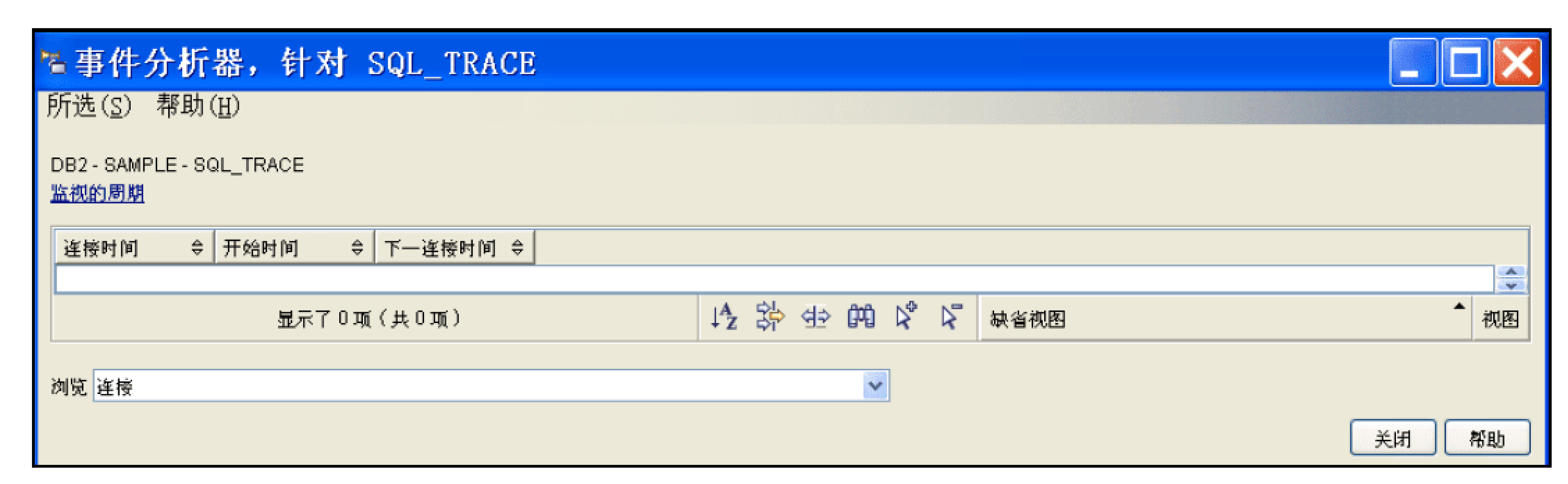


图 4-17　事件分析器

4.3　DB2 CLP 处理程序

4.3.1　DB2 CLP 简介

DB2 命令行处理器听起来没什么特别之处，但实际上它是 DB2 的接口，它充分体现了 DB2 的威力，以及 DB2 的简单性和通用性。命令行使我们想起了 UNIX 上的 Telnet，还有我们常使用的 DOS 命令。DB2 Command Line Processor(DB2 CLP)是所有 DB2 产品都必备的工具，可以使用这个应用程序运行 DB2 命令、操作系统命令或 SQL 语句。虽然您的最终用户可能永远不会使用 CLP 来访问他们的数据，但是对于 DBA 或者应用程序的编程人员来说，CLP 在工具箱中则是最基本的工具——它就像是在所有场合都适用的燕尾服。数据库通常是核心应用最关键的组成部分，如果数据库层面发生故障，问题的判定、解决相对会比较困难。而 CLP 正是 DBA 访问数据库进而诊断、排除故障的一个入口。

注意：

其实每个数据库都提供有类似 DB2 CLP 的命令行接口工具，例如 Oracle 的 SQL*PLUS；Informix 的 DBACCESS；Sybase 的 isql。

4.3.2　DB2 CLP 设计

CLP 从架构上来说由两个过程组成：

- 一个前端进程(或者是在 Windows 上的线程)，用于处理与操作系统命令提示符的通信。
- 一个后端进程，用于处理与数据库的通信。这确保了在您连接到 DB2 之后，如果用 Control-C 或 Control-Break 中止来自一个大型选择(例如 SELECT* FROM SYSCAT.TABLES)的输出，那么会顺利中止输出，而不会断开与 DB2 的连接。

命令窗口与 Windows 上的 CLP 的比较

在 Linux 和 UNIX 环境下，我们可以通过命令行界面直接输入 db2 操作命令来调用 DB2 CLP；而在 Windows 环境下则不行，需要在命令窗口先执行 db2cmd 命令或者 db2cw 命令启动 CLP。为什么在 Windows 环境下，DB2 要求您用 DB2CMD.EXE 启动一个 CLP 呢？这是因为在 UNIX 和 Linux 上，前端进程和后端进程之间的连接非常简单：如果父进程死亡，则其所有子进程都将被操作系统终止。而在 Windows 上，父线程在死亡时并不会终止其子线程。因此，DB2 使用一个 cookie 来为 Windows 上的 CLP 链接前端线程和后端线程。这就要求 CLP 必须通过 DB2CMD.EXE 来启动。这样做可以确保如果杀死了父线程，那么子线程也不会保留下来，从而避免了资源的浪费。如果 DB2 不采用这种技术，就会产生大量的 phantom 线程。

DB2 中有两种不同的处理器：DB2 命令行处理器(DB2 CLP)和 DB2 命令窗口(DB2 CW)。有人喜欢用同样的名字 DB2CLP 称呼它们，因为它们在 Windows“开始”菜单中有相同的图标。对于除 Windows 之外的所有操作系统，DB2 CW 是在操作系统的本机 CLP 中内置的。在 Windows 环境中，可以在 Windows 命令提示中输入 db2cmd 命令或者 db2cw 命令来启动 DB2CW，或者还可以在 DB2 的“命令行工具”菜单中选择“命令窗口”来启动 DB2 CW；而 DB2 命令行处理器则可以通过在 DB2 的“命令行工具”菜单中选择“命令行处理器”来启动，或者在 DB2 命令窗口中输入 db2 命令来启动，这时就进入到交互模式，这种模式称作 DB2 交互式 CLP，它会创建如下所示的一个特殊的提示符：

```
db2 =>
```

在 DB2 交互式 CLP 下，您就不必在执行 DB2 命令时加上 DB2 前缀。但这时您必须在执行操作系统命令时加上一个惊叹号(!)前缀。例如，如果想运行 dir 命令，就必须输入 !dir。

图 4-18 显示通过 DB2 CW 以非交互式的方式输入的一个命令。

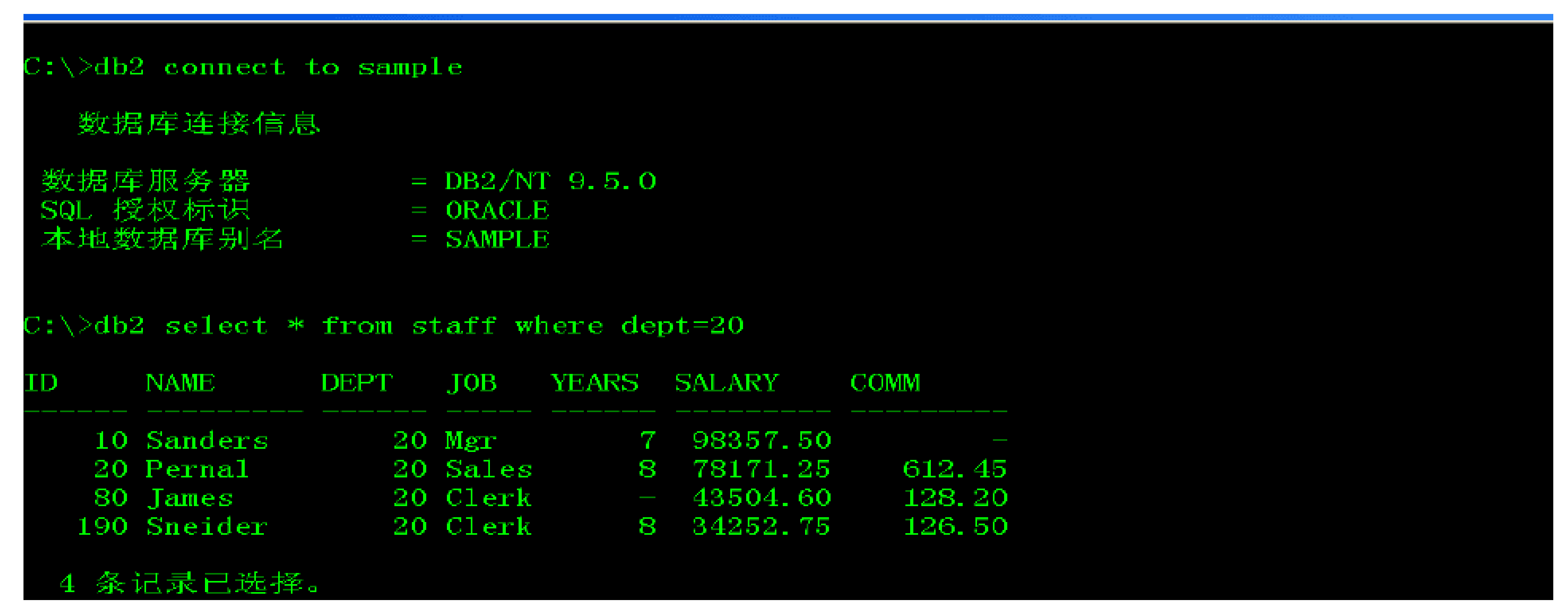


图 4-18　用 DB2 CW 输入命令

注意，运行这个 DB2 命令时，必须输入关键字 db2 前缀。如果不这么做，操作系统就会认为这是一个操作系统命令，会返回错误。如果使用 DB2 交互式 CLP，就不需要这么做，如图 4-19 所示。

```
C:\>db2
(c) Copyright IBM Corporation 1993,2007
DB2 客户机 9.5.0 的命令行处理器

可从命令提示符处发出数据库管理器命令和 SQL 语句。例如：
    db2 => connect to sample
    db2 => bind sample.bnd

 (Instance:%I, Database: ):connect to sample

   数据库连接信息

 数据库服务器          = DB2/NT 9.5.0
 SQL 授权标识          = ORACLE
 本地数据库别名        = SAMPLE

 (Instance:%I, Database: SAMPLE):
```

图 4-19　用 DB2 CW 以交互模式输入命令

4.3.3　DB2 CLP 命令选项

在使用 DB2 CLP 处理程序时，可以使用命令行选项修改处理过程或处理中输入的语句和命令的行为方式。在调用 DB2 命令时，可以指定一个或多个处理程序选项。常用的一些选项如下：

- 可以使用 c 标志定义每个语句的自动提交；
- 可以使用 v 标志定义在命令执行时同时在屏幕输出；
- 可以使用 s 标志定义执行命令序列时碰到错误停止执行；
- 可以使用 z 标志定义把命令执行期间的结果输出到一个文件；
- 可以使用 f 标记定义提供 DB2 命令和 SQL 语句的输入文件；
- 可以使用 t 标记定义语句末尾的结束字符(默认字符是“;”)。如果你不想用“;”结尾，而想用@结尾，那么可以使用-td@方式读取输入文件。

可以通过在 DB2 处理程序中输入 db2 list command options 命令或者 db2 ? options 命令获得所有有效选项的列表。如图 4-20 所示，运行这个命令，会看到 19 个以上的选项。

有 3 种修改 DB2 处理程序选项的方法：

- 可以通过 db2set DB2OPTIONS 直接修改注册表变量，这会永久改变 DB2 处理程序选项的值。例如：

```
C:\>db2set db2options=-c
```

```
C:\>db2 list command options

    命令行处理器选项设置

 后端进程等待时间（秒）          (DB2BQTIME) = 1
 连接至后端的重试次数             (DB2BQTRY) = 60
 请求队列等待时间（秒）          (DB2RQTIME) = 5
 输入队列等待时间（秒）          (DB2IQTIME) = 5
 命令选项                       (DB2OPTIONS) =

  选项   描述                                       当前设置
 ------ ----------------------------------------  ---------------
   -a    显示 SQLCA                                 OFF
   -c    自动落实                                   ON
   -d    检索并显示 XML 声明                        OFF
   -e    显示 SQLCODE/SQLSTATE                      OFF
   -f    读取输入文件                               OFF
   -i    显示 XML 数据并带有缩进                    OFF
   -l    将命令记录到历史记录文件中                 OFF
   -m    显示受影响的行数                           OFF
   -n    除去换行字符                               OFF
   -o    显示输出                                   ON
   -p    显示交互式输入提示符                       ON
   -q    保留空格和换行符                           OFF
   -r    将输出保存到报告文件                       OFF
   -s    在命令出错时停止执行                       OFF
   -t    设置语句终止字符                           OFF
   -v    回传当前命令                               OFF
   -w    显示 FETCH/SELECT 警告消息                 ON
   -x    不打印列标题                               OFF
   -z    将所有输出保存到输出文件                   OFF
```

图 4-20　各种 DB2 CLP 选项

- 可以使用 update command options 命令修改 DB2 处理程序选项，这会在会话级改变 DB2 处理程序选项的值。它的优先级高于使用 DB2set DB2OPTIONS 设置，它将会覆盖在注册表级建立的任何设置。例如：

```
db2=>update command options using c on
DB20000I  UPDATE COMMAND OPTIONS 命令成功完成。
db2=>
```

- 在输入 DB2 命令时指定命令行标志，这将会在语句级改变 DB2 处理程序选项的值。它的优先级高于使用 DB2set DB2OPTIONS 设置，也高于使用 update command options 命令设置，它将会覆盖注册表级和会话级建立的所有设置。例如：

```
db2 -c command or statement...
```

正如上面的图 4-20 中所示，在使用指定命令行标志打开选项时，应该在对应的选项字母前面加上减号(–)；例如，要打开自动提交特性(这是默认的)。

要关闭选项，可以在选项字母前后加上减号(-c-)，或者在前面加上加号(+)。再解释一下前两句话，因为这儿可能有点儿混乱：在标志前面放上减号会打开选项；在标志前面和

后面都放上减号，或者在标志前面放上加号会关闭选项。这么说也许还是不太明确，因为这可能导致混淆，我们用自动提交选项举个例子。

一些命令行选项是默认打开的，其他的是默认关闭的。前面的解释(和后面的示例)描述默认打开的命令行选项的行为和效果。如果命令行选项是默认关闭的，那么使用相反的逻辑。在默认情况下，自动提交特性是打开的(-c)。这个选项指定每个语句是否自动提交或回滚。

如果一个语句成功了，它就和它前面执行的未提交的(关闭了自动提交所致)的所有成功语句一起提交。但是，如果它失败了，它就和它前面执行的未提交的(关闭了自动提交所致)所有成功语句一起回滚。如果这个语句关闭了自动提交，就必须显式地执行提交或回滚命令。

在图 4-21 中，在命令行上修改了自动提交特性的值来演示这个过程。



图 4-21　在运行时修改命令行选项

那么，发生了什么情况？首先，我创建了一个称为 A 的表，但是在执行这个任务时使用+c 选项关闭了默认的自动提交选项(也可以在这个标志前后加上减号[-c-]，效果是一样的)。在创建表 A 之后(请记住，没有提交这个操作)，我创建了另一个称为 B 的表，这一次也关闭了自动提交特性。然后对这两个表进行 Cartesian 联结，同样动态地关闭 DB2 CLP 的自动提交特性。最后，做一次回滚并再次运行同样的 SELECT 语句，这一次这个语句失败了。

如果您看看这个事务，就会发现我没有执行提交操作。如果第一个 SELECT 没有包含+c 选项，那么就会提交创建表 A 和 B 的结果(因为这个 SELECT 成功了)；因此后面的回滚不会影响这两个已经提交的表，第二个 SELECT 语句会成功地返回与第一个 SELECT 语句相同的结果。

试一下相同的命令序列，但是这一次使用-c-选项。应该会看到同样的结果。在此之后，再试一次，这一次在第一个 SELECT 语句中不使用任何选项，看看第二个 SELECT 语句是否会返回结果。

您的操作系统可能对在一个语句中可以读取的最大字符数量有限制(即使命令行在显示器上转入下一行)。为了在输入长语句时解决这个限制，可以使用续行字符(\)。当 DB2 遇到续行字符时，它读取下一行并在处理时将两行合并。在两种 DB2 处理程序中都可以使用这个字符。但是，要知道 DB2 对一个语句的限制是 2MB(这对于命令行应该足够了)。图 4-22 演示了它在 DB2 CLP 中的使用方法。

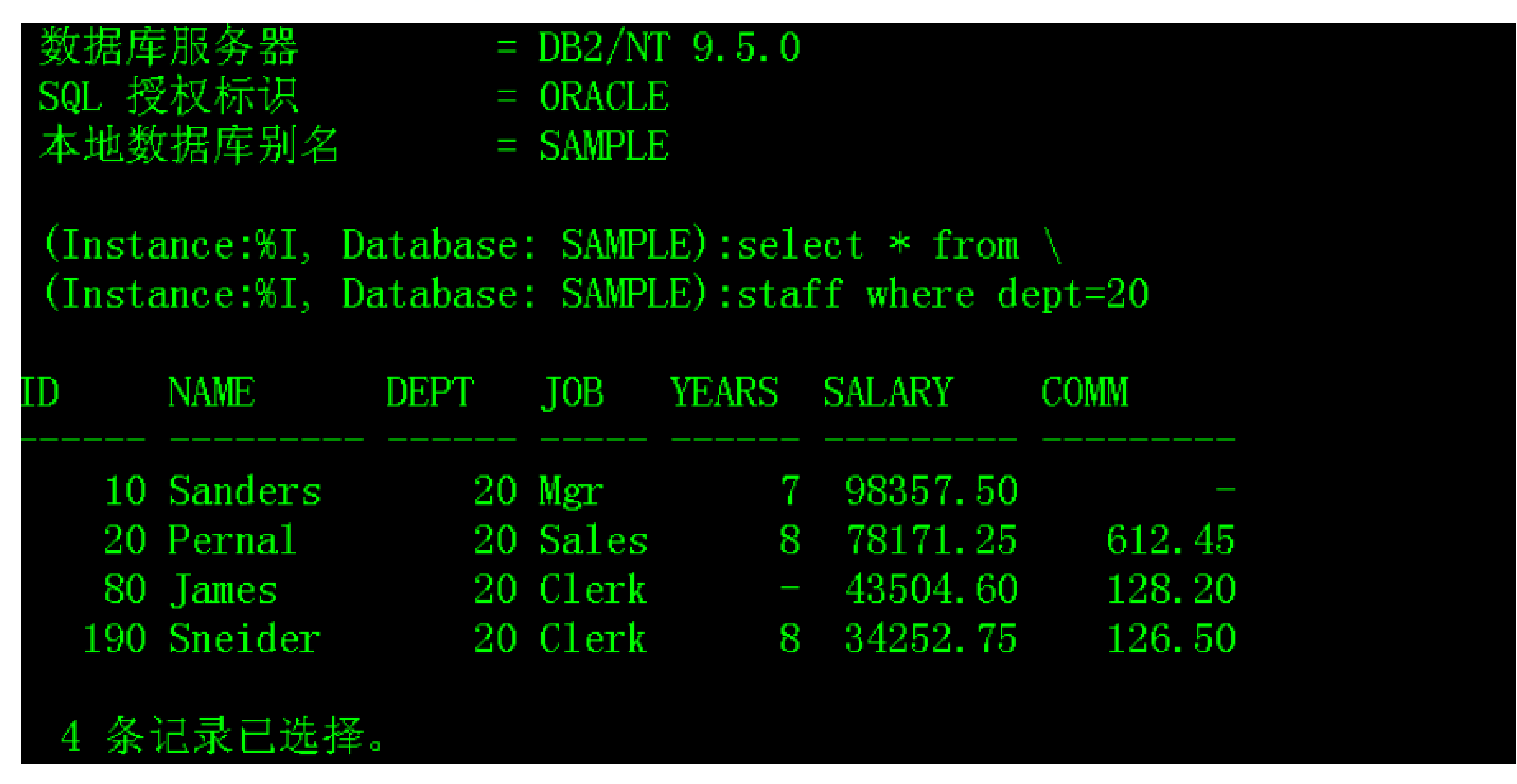


图 4-22　在 DB2 CLP 中使用续行字符

如果使用 DB2 CW 输入命令，那么下面这些特殊字符会导致问题：

```
$ & *() ; < > ? \ ' "
```

操作系统 shell 可能会错误地解释这些字符(当然，在 DB2 CLP 中不存在这个问题，因为它是为 DB2 命令专门设计的应用程序)。

可以将整个语句或命令放在引号中，从而表示希望由 DB2 解释系统操作符，而不是由操作系统进行解释，如下所示：

```
db2 "select * from staff where dept > 10"
```

试着在 DB2 CW 中输入相同的命令，但是不加引号。会发生什么？查看发出这个命令

时您所在目录的内容，您一定会找到一个称为 10 的文件，其中包含一个 SQL 错误。为什么呢？DB2 解释 SQL 语句

```
select * from staff where dept
```

并将产生的内容放进文件 10。“>” 符号是一个操作系统指令，表示将来自标准显示的输出管道连接到指定的文件(在这个示例中，是 10)。select * from staff where dept 语句当然是一个不完整的 SQL 语句，因此会产生错误。不正确的结果是由于操作系统错误地解释了特殊字符。

查看 DB2 命令帮助信息，如果想在 CLP 中查看命令帮助信息，可以输入 db2 ?选项，如图 4-23 所示。

```
db2 ?
db2 ? command string
db2 ? SQLnnnn    (nnnn = 4 or 5 digit SQLCODE)
db2 ? nnnnn      (nnnnn = 5 digit SQLSTATE)
```

图 4-23　查看 DB2 命令帮助信息

4.3.4　设置 DB2_CLPPROMPT 定制 DB2 CLP

DB2 提供了两种从命令行界面输入命令的方式。当以交互(Interactive)模式使用 DB2 命令行处理器(DB2 Command Line Processor，DB2 CLP)时，您不必在 DB2 命令或 SQL 查询前加上关键字 DB2。

请看一下图 4-24，图中在运行于交互模式下的 DB2 UDB CLP 中输入了 SELECT*... 语句。您知道这个特定表(STAFF)位于哪个数据库或实例吗？您大概不知道吧(虽然对于这个特例，您可以猜测)；可是，DB2 知道！

```
db2 => connect to sample

   数据库连接信息

 数据库服务器          = DB2/NT 9.5.0
 SQL 授权标识          = ORACLE
 本地数据库别名        = SAMPLE

db2 => select * from t1 where id=1

ID
-----------
          1

  1 条记录已选择。
```

图 4-24　默认的 CLP 视图不会告诉您连接到了哪个实例

现在看一下图 4-25 中一模一样的查询。您现在能回答我的问题了吗？注意到有什么不同了吗？

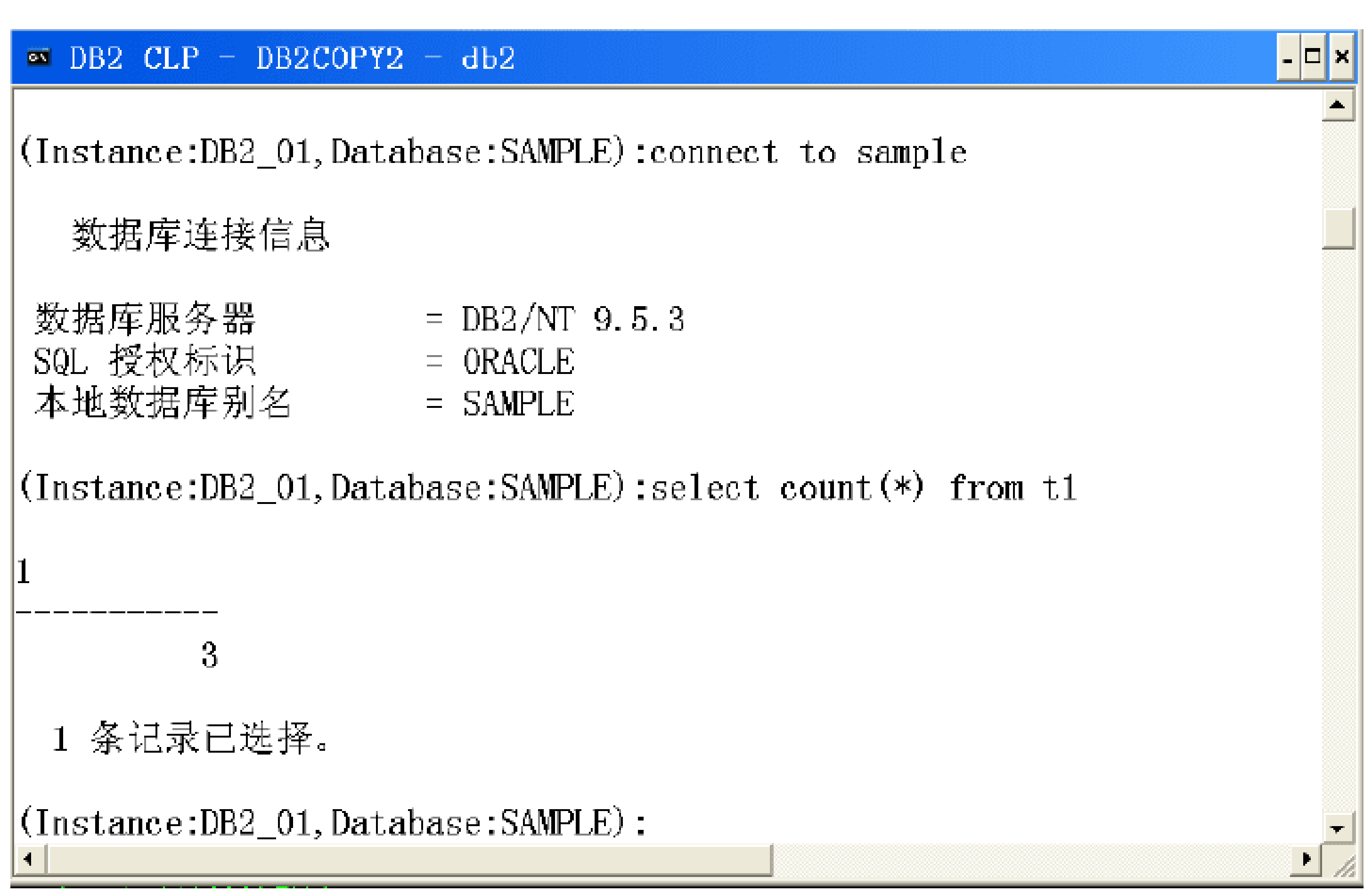


图 4-25 定制 CLP 以显示您所连接的实例和数据库

DB2 通过设置注册变量可以让您在运行于交互模式时，定制 DB2 UDB CLP 提示符(后文所指的 DB2 CLP 就是运行于交互模式的 DB2 CLP)。

您可以使用这项新的定制功能，把您自己的文本和反映当前实例连接(instance attachment)和/或数据库连接(database connection)的上下文(context)的变量添加到输出显示。下面我们介绍这项新的定制功能。

定制交互模式下的 DB2 CLP

现在 DB2 UDB CLP 提示符可以反映当前实例连接和数据库连接的上下文，还可以显示特定的字符消息。如果没有该项功能，使用交互模式下的 DB2 CLP 会显示硬编码的提示符，如图 4-26 所示。

设置 DB2_CLPPROMPT 注册表变量

要定制 DB2 CLP 命令提示符，请使用新的 DB2 注册表变量——DB2_CLPPROMPT。

可使用 db2set 命令更新 DB2 注册表变量(注：关于 DB2 注册表变量和 db2set 的详细信息我们已在第 2 章中讲解过了)，这些信息被立即存储到实例注册变量中。DB2 实例注册变量将这些更新过的信息应用到在进行更改之后启动的 DB2 服务器实例和应用程序。如果需要永久性设置某个环境变量，那么您应该使用 db2set 命令在 DB2 服务器的启动实例注册变量内设置它。而 db2set 命令行则将环境变量永久性地设置在 DB2 实例注册变量中。



图 4-26　CLP 中的硬编码提示符

要查看全部受支持的注册表变量列表，请输入以下命令：

```
db2set -lr
```

要更改 DB2 UDB 注册表变量的值，请输入以下命令：

```
db2set registry_variable_name=new_value
```

要查看被设置的全部 DB2 注册表变量列表，请输入以下命令：

```
db2set -all
```

您可以将 DB2_CLPPROMPT 设置为长度不超过 100 个字符的任何文本字符串。这个定制的字符串可包含在运行时可替换的可选标记。如果这个注册表变量在 DB2 CLP 会话期间发生更改，那么新的值在用户退出再重新进入该处理器后方可生效。

可以将 DB2 UDB CLP 定制为只显示一行字符串，这是最基本的形式。图 4-27 和图 4-28 演示了 DB2_CLPPROMPT 注册表变量的设置以及 DB2 UDB CLP 的后续调用。

```
C:\>db2set DB2_CLPPROMPT="你正在使用DB2 V9.5,请输入命令: "

C:\>db2cmd db2

C:\>
```

图 4-27　设置命令行提示符注册表变量

```
(c) Copyright IBM Corporation 1993,2007
DB2 客户机 9.5.0 的命令行处理器

可从命令提示符处发出数据库管理器命令和 SQL 语句。例如:
    db2 => connect to sample
    db2 => bind sample.bnd

要获得一般帮助，输入：?。
要获得命令帮助，输入：? command，其中 command 可以是
数据库管理器命令的前几个关键字。例如:
 ? CATALOG DATABASE 用于关于 CATALOG DATABASE 命令的帮助
 ? CATALOG          用于关于所有 CATALOG 命令的帮助。

要退出 db2 交互方式，在命令提示符处输入
QUIT。在非交互方式下，所有命令都必须以“db2”作前缀。
要列出当前命令选项设置，输入 LIST COMMAND OPTIONS。

要获取更详细的帮助，请参阅 Online Reference Manual。

你正在使用DB2 V9.5,请输入命令:
```

图 4-28　结果

现在，尽管这个示例挺有意思，但没什么大用。DB2_CLPPROMPT 注册表变量可以附带一些关联变量，可以用这些变量向 DB2 CLP 返回一些关于当前的或默认的实例连接，以及当前所连接的(或默认的)数据库的信息。

受支持的变量如表 4-1 所示。

表 4-1　受 CLP 支持的运行时变量

变　量	运 行 时 值
%ia	如果存在实例连接，则为当前实例连接的授权标识(authid)；否则为空字符串
%i	如果实例连接存在，则为当前所连接的实例的本地别名；如果不存在本地实例连接，则为 DB2INSTANCE 或 DB2INSTDEF 注册表变量的值；否则，为空字符串
%da	如果存在数据库连接，则为当前数据库连接的授权标识；否则为空字符串
%d	如果数据库连接存在，则为当前连接的数据库的本地别名；否则为 DB2DBDFT 注册表变量值；再不然则为空值
%n	换行符

例如，要设置 DB2 UDB CLP 提示符，使其解析为:

```
 (Instance <instance_name>, Database <database name>):
```

输入以下命令:

```
db2set db2_clpprompt=" (Instance:%i, Database: %d):"
```

您可以输入 db2set -all 命令来验证 DB2 UDB 概要注册表中的该项设置。

图 4-29 向您显示了这一命令序列，包括在以交互模式启动 CLP 会话之后的显示结果。

```
C:\>db2set DB2_CLPPROMPT="你正在使用DB2 V9.5,请输入命令: "

C:\>db2cmd db2

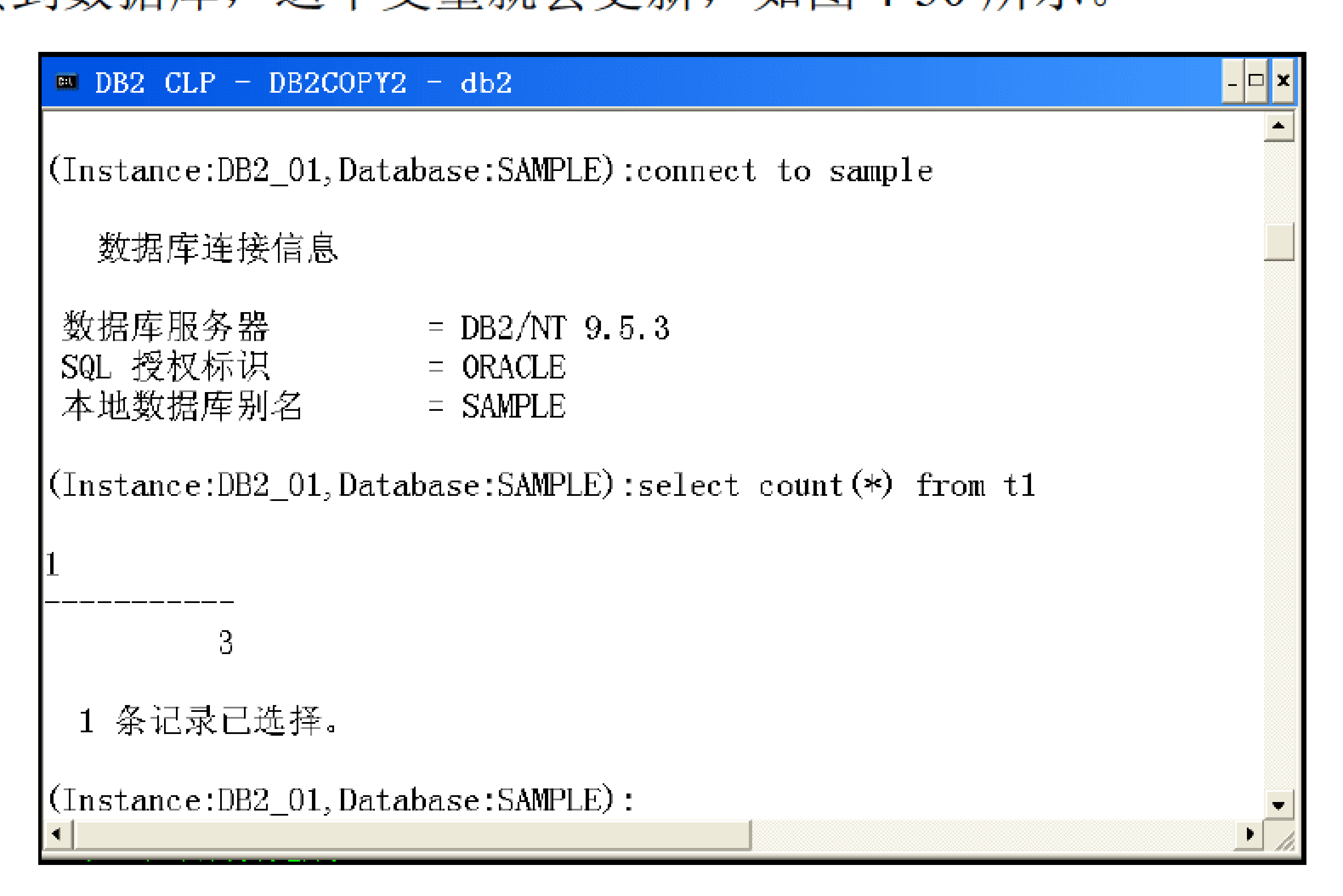
C:\>db2set -all
[e] DB2PATH=C:\Program Files\IBM\SQLLIB
[i] DB2_CLPPROMPT=你正在使用DB2 V9.5,请输入命令:
[i] DB2ACCOUNTNAME=XINZHUANG\oracle
[i] DB2INSTOWNER=XINZHUANG
[i] DB2PORTRANGE=60000:60003
[i] DB2INSTPROF=C:\DOCUMENTS AND SETTINGS\ALL USERS\APPLICATION DATA\IBM\DB2\DB2COPY1
[i] DB2COMM=TCPIP
[g] DB2_EXTSECURITY=YES
[g] DB2SYSTEM=XINZHUANG
[g] DB2PATH=C:\Program Files\IBM\SQLLIB
[g] DB2INSTDEF=DB2
[g] DB2ADMINSERVER=DB2DAS00
```

图 4-29　设置 DB2_CLPPROMPT 并验证其使用

请注意此例中，我用了未处于交互模式的 DB2 CLP，这就是为什么交互模式在同一个窗口中被启动的原因(我这样做是让您体会一下调用 DB2 CLP 的不同方式)。

图 4-29 中，您会看到<*database_name*>变量没有值。该变量之所以为空是因为在我的环境中没有数据库连接或者定义的默认数据库。

如果您连接到数据库，这个变量就会更新，如图 4-30 所示。



```
DB2 CLP - DB2COPY2 - db2

(Instance:DB2_01,Database:SAMPLE):connect to sample

   数据库连接信息

 数据库服务器         = DB2/NT 9.5.3
 SQL 授权标识         = ORACLE
 本地数据库别名       = SAMPLE

(Instance:DB2_01,Database:SAMPLE):select count(*) from t1

1
-----------
          3

  1 条记录已选择。

(Instance:DB2_01,Database:SAMPLE):
```

图 4-30　出现在提示符上的实例名和数据库名

如果我从这个数据库断开，这个定制的字符串会反映出这一操作。如果我接着再连接到不同的实例，在 DB2 CLP 中也会得到动态的反映，如图 4-31 所示。

```
(INSTANCE:DB2_01,DATABASE:):attach to db2inst1

   实例连接信息

 实例服务器             = DB2/NT 9.5.0
 授权标识                           = ORACLE
 本地实例别名           = DB2INST1

(INSTANCE:DB2INST1,DATABASE:):
```

图 4-31 连接到新的 DB2 实例时，CLP 动态进行更新

4.4 配置 DB2 服务器的 TCP/IP 通信

客户端要想访问 DB2 数据库服务器，必须先配置 DB2 服务器上的通信协议，DB2 服务器才会接受来自远程 DB2 客户机的建立连接请求。

在配置 DB2 实例的 TCP/IP 通信之前，必须检查以下内容：

- DB2 服务器正在使用 TCP/IP，则 DB2 客户机也必须正在使用 TCP/IP 才能建立连接；
- 标识“连接服务名称”和“连接端口”，或仅标识“连接端口”。

连接服务名称和连接端口

该服务名称用于更新服务器上的数据库管理器配置文件中“服务名称”(svcename)参数。当指定“连接服务名称”时，必须以相同的“服务名称”、端口号和协议更新 services 文件，services 文件包含在服务器上定义的服务及其端口号。“服务名称”是任意的，但是在 services 文件内必须是唯一的。服务名称的样本值可以是 server1。“连接端口”在 services 文件中必须是唯一的。端口号和协议的样本值可以是 50000/tcp。

连接端口

可以选择不使用“连接服务名称”而只是使用“连接端口号”更新服务器上的数据库管理器配置文件中“服务名称”(svcename)参数。这时，就不会用到 services 文件，自然也不必更新 services 文件。如果正在使用分区格式的“DB2 企业服务器版”，那么必须确保端口号与“快速通信管理器”(FCM)或系统上的任何其他应用程序使用的端口号没有冲突。端口号的样本值可以是 50000。

要配置 DB2 实例的 TCP/IP 服务器通信，需要以下几个步骤。

4.4.1　在服务器上更新 services 文件

TCP/IP services 文件指定服务器应用程序侦听客户机请求的端口。如果在 DBM 配置文件的 svcename 字段中指定了服务名称，那么必须在 services 文件中添加一行，写入服务名称与端口号/协议的映射关系。如果在 DBM 配置文件的 svcename 字段中指定的不是服务名称而直接是端口号，则不需要更新 services 文件。

在这里需要指出，services 文件的默认位置取决于操作系统，参考表 4-2。

表 4-2　services 文件的位置

操 作 系 统	目　　录
Windows	%SystemRoot%\system32\drivers\etc，其中%SystemRoot%是系统定义的环境变量
Linux 或 UNIX	/etc

使用文本编辑器将“连接”条目添加至 services 文件。例如：

```
db2c_db2inst1  50000/tcp #DB2 连接服务端口
```

其中：

- db2c_db2inst1 表示连接服务名称
- 50000 表示连接端口号(#50000 是 DB2 实例的默认端口)，读者可以根据自己需要更改
- tcp 表示您使用的通信协议

4.4.2　在服务器上更新数据库管理器配置文件

更新数据库管理器配置文件在配置 DB2 实例的 TCP/IP 通信过程中是必不可少的一环。必须用服务名称(svcename)参数更新数据库管理器配置文件。

要更新数据库管理器配置文件，必须完成以下操作：

(1) 作为具有“系统管理员”(SYSADM)权限的用户登录系统。

(2) 启动 DB2 命令行处理器(CLP)。

(3) 通过输入下列命令，用“服务名称”(svcename)参数更新数据库管理器配置文件：

```
db2 update database manager configuration using svcename
```

```
[service_name|port_number]
          db2stop
          db2start
```

其中：

service_name 是 services 文件中保留的服务名称

port_number 是 service_name 的相应端口号或空闲的端口号(如果未保留 service_name)，如果正在指定服务名称，那么使用的 svcename 必须与在 services 文件中指定的“连接服务名称”相匹配。

这里需要注意，svcename 不能联机配置，必须停启实例后才能生效。

在停止并再次启动数据库管理器之后，查看数据库管理器配置文件以确保这些更改已经生效。通过输入下列命令，查看数据库管理器配置文件：

```
    db2 get database manager configuration | find /i “svcename”
```

4.4.3 设置 DB2 服务器的通信协议

要执行此任务，需要 sysadm 权限。为 DB2 实例设置通信协议是为 DB2 实例配置 TCP/IP 或 SSL 通信的主要任务的一部分。

DB2COMM 注册表变量允许您设置当前 DB2 实例的通信协议。如果 DB2COMM 注册表变量未定义或设置为空，那么启动数据库管理器时不会启动任何协议连接管理器。

可以使用下列其中一个关键字来设置 DB2COMM 注册表变量：tcpip 启动 TCP/IP 支持，ssl 启动 SSL 支持。

要为实例设置通信协议：

从 DB2 命令窗口输入 db2set DB2COMM 命令：

```
db2set DB2COMM=tcpip
```

例如，要将数据库管理器设置为对 TCP/IP 通信协议启动连接管理器，输入以下命令：

```
db2set DB2COMM=tcpip
db2stop
db2start
```

4.4.4 查看服务器通信端口状态

在做完上面 3 个步骤后，在系统输入 netstat 来查看通信端口的状态，如图 4-32 所示。通信端口必须处于“LISTENING”状态才能侦听来自客户端的 tcp 请求。

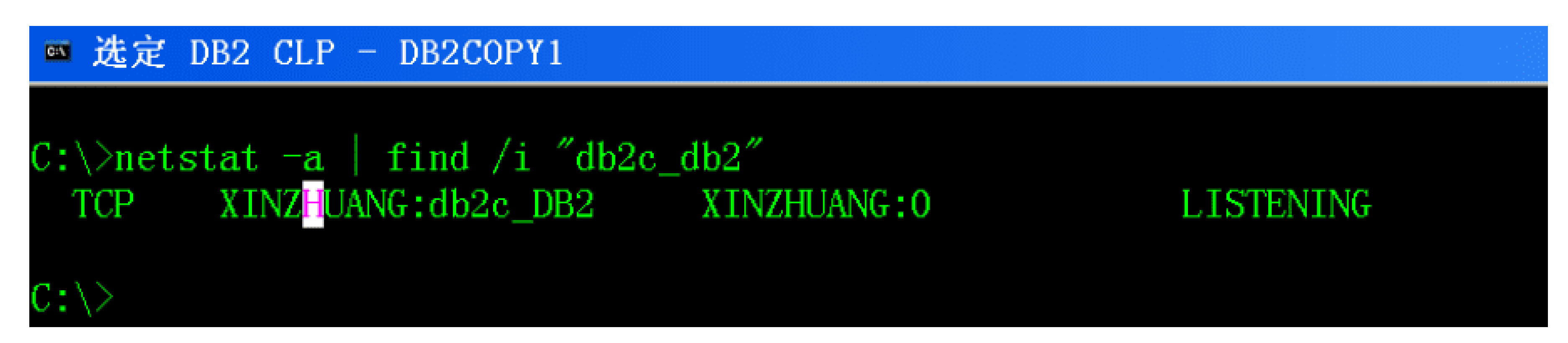


图 4-32　查看通信端口的状态

4.4.5　使用控制中心配置 DB2 服务器通信

控制中心的设置通信功能允许我们显示一个服务器实例在配置之后可使用的协议和配置参数。它还允许您修改已配置协议的参数值，也允许您添加或删除协议。

向服务器系统添加对新协议的支持时，设置通信功能检测并生成新协议的服务器实例参数值。在使用之前，可接受或修改这些值。当从服务器系统中除去对现存协议的支持时，设置通信功能检测已除去的协议，并禁止该服务器实例使用它。

可添加尚未检测到的协议，但是，在继续执行之前必须提供所有必需的参数值。设置通信窗口如图 4-33 所示。

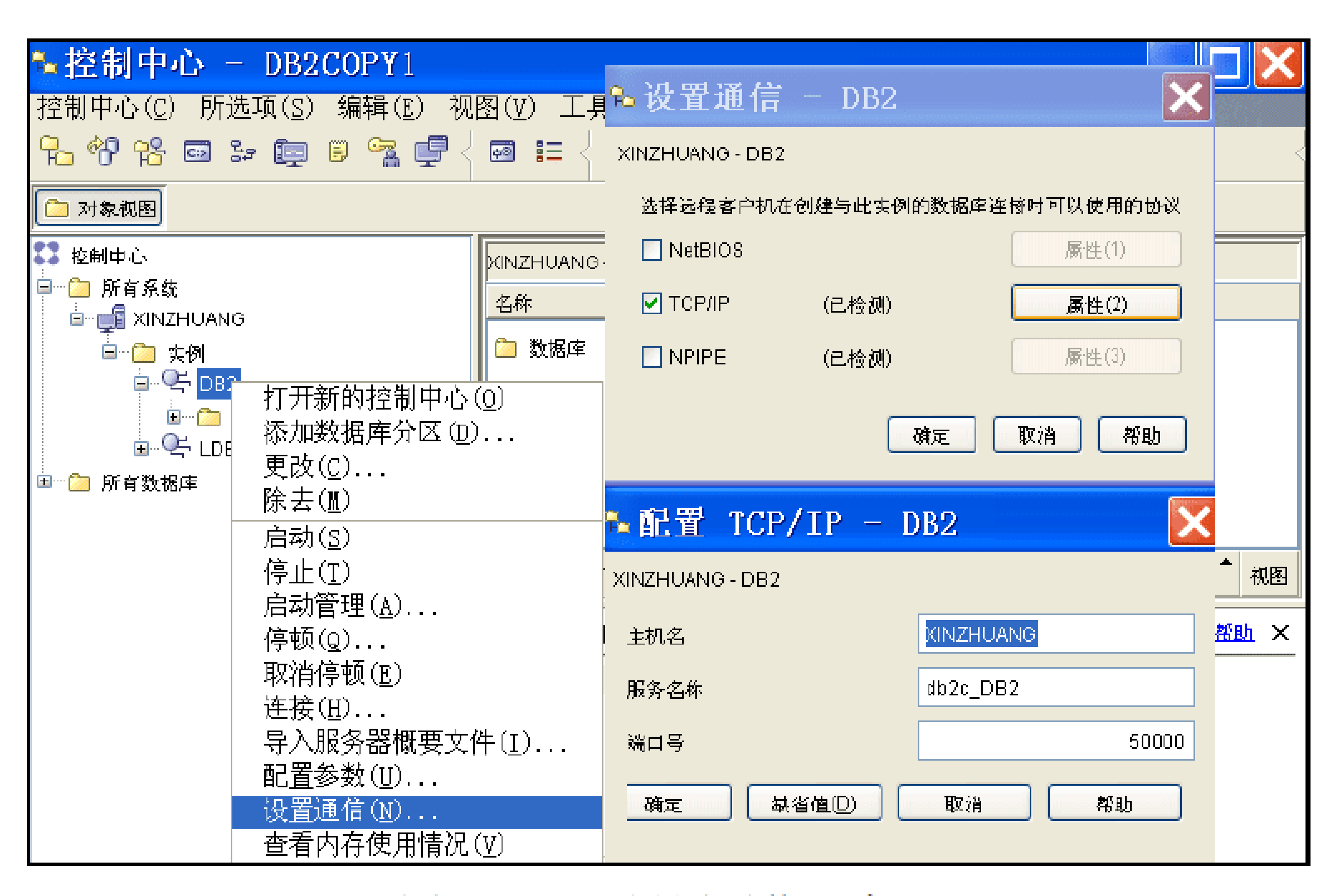


图 4-33　“设置通信”窗口

设置通信功能可用于维护本地和远程服务器实例的通信(要配置远程实例的 DB2 通信协议，必须要求远程 DB2 实例所在的服务器和本地服务器启动管理服务器(DAS))。

要修改先前已经配置的实例通信设置，可能需要更新客户机上的数据库连接目录。为此，可以：

- 在客户机上使用配置助手 CA。选择想要更改的数据库连接。在“所选项”菜单下，选择“更改数据库”。这将启动“向导”，它将帮助您进行更改。

- 根据服务器上已更改的值，在客户机上使用命令行处理器来对节点取消编目和重新编目。

4.5 配置客户机至服务器通信

4.5.1 客户机至服务器通信概述

要想配置客户机至服务器通信，必须先了解客户机至服务器通信有关的基本组件：

- **客户机**——指的是通信的发起方。
- **服务器**——指的是来自客户机的通信请求的接收方。
- **通信协议**——指用来在客户机和服务器之间发送数据的协议。DB2 产品支持以下几个协议：
  - ◊ TCP/IP。可根据版本进行更进一步的区分：TCP/IPv4 或 TCP/IPv6。
  - ◊ IPC(进程间通信)。此协议用于本地连接。

客户机至服务器通信：连接类型

通常，提到设置客户机至服务器通信时指的是远程连接，而不是本地连接。

本地连接是一个数据库管理器实例与由那个实例管理的数据库之间的连接。换句话说，CONNECT 语句从数据库管理器实例发出给它自己。本地连接是独特的，因为不需要设置通信并且使用了 IPC。

远程连接是一个在其中发出 CONNECT 语句到数据库的客户机和数据库服务器处于不同位置的连接。通常，客户机和服务器在不同的机器上。然而，如果客户机和服务器在不同的实例中，那么远程连接可能存在于同一台机器上。

另一个较不常用的连接类型是回送连接(loopback)。这是一种远程连接类型，该连接配置为从一个 DB2 实例(客户机)到相同的 DB2 实例(服务器)。

可以通过控制中心、CA 配置助手和 CLP 来配置客户机到服务器通信，下面我们分别讲解这 3 种不同的配置方式。

4.5.2 使用控制中心配置客户端通信

使用控制中心配置远程客户端和服务器通信，要求远程 DB2 实例所在的服务器和本地服务器必须启动管理服务器(DAS)，在远程和本地服务器上启动管理服务器后，执行下列步骤：

(1) 启动控制中心。

(2) 如果列出了包含您想要的远程实例的系统，那么单击系统名称旁边的[+]号显示“实例”文件夹。单击“实例”文件夹旁边的[+]，以显示该系统上所有实例的列表，然后转至步骤(13)。如果已列出包含您想要的远程实例的系统，但您所要的实例未出现在该系统下面，那么转至步骤(8)。

(3) 如果未列出包含您想要配置的远程实例的系统，那么选择**系统**文件夹，单击鼠标右键并选择**添加**选项。“添加系统”窗口将打开。

(4) 要向控制中心添加系统，可执行下列其中一项操作：

- 如果系统名称为空，那么单击 **Discover** 以显示网络上的 TCP/IP 系统的列表。选择一个系统并按**确定**。在“添加系统”窗口上填充系统信息。
- 如果已填写系统名称，那么单击 **Discover** 以调用已知 discovery。如果成功，那么在“添加系统”窗口上填充系统信息。

注意：

Discovery 只对远程 TCP/IP 系统起作用。

(5) 单击**应用**以将系统添加至控制中心窗口。

(6) 单击**关闭**。

(7) 单击您刚刚添加的系统名称旁边的[+]号以显示“实例”文件夹。

(8) 为新系统选择**实例**文件夹并单击鼠标右键。

(9) 选择**添加**选项。“添加实例”窗口将打开。

(10) 单击 **Discover** 以获取可用实例的列表并在系统上显示远程实例的列表。

(11) 选择想要添加的实例并单击**确定**。“添加实例”窗口将填充远程实例信息。

(12) 单击**关闭**。

(13) 选择要配置的实例并单击鼠标右键。

(14) 从弹出菜单中选择“设置通信”选项。“设置通信”窗口将打开。

(15) 使用“设置通信”窗口为该实例配置通信协议。

(16) 必须停止该实例，然后再重新启动它，才可使这些更改生效：

- 要停止一个实例，选择该实例，单击鼠标右键，并选择**停止**选项。
- 要启动一个实例，选择该实例，单击鼠标右键，并选择**启动**选项。

4.5.3 使用 CA 配置客户机到服务器通信

客户机配置助手(CA)图形化工具能够非常快速轻松地帮助我们配置客户端到服务器的通信。下面的这个案例中我们使用 CA 对一个远程 DB2 服务器上的数据库进行编目。对数据库进行编目之后，就能够像访问本地数据库一样访问远程数据库。DB2 会在“幕后”

执行所有通信过程。

配置步骤

(1) 熟悉以下远程数据库信息：

通信协议(PR) Protocol = TCPIP

IP 地址(IP) IP Address or hostname = 192.168.1.1

端口号(PN) Instance Port Number = 50000

数据库名称(DB) Database Name = SAMPLE

提示：

在 Windows 上获取主机名的方法是在命令窗口中输入 *hostname*

在 Windows 上获取 IP 地址的方法是在命令窗口中输入 *ipconfig*

(2) 打开“配置助手”(提示：可以通过“开始”菜单访问它)。

从开始菜单中找到 IBMDB2 的配置助手，单击启动它，如图 4-34 所示。

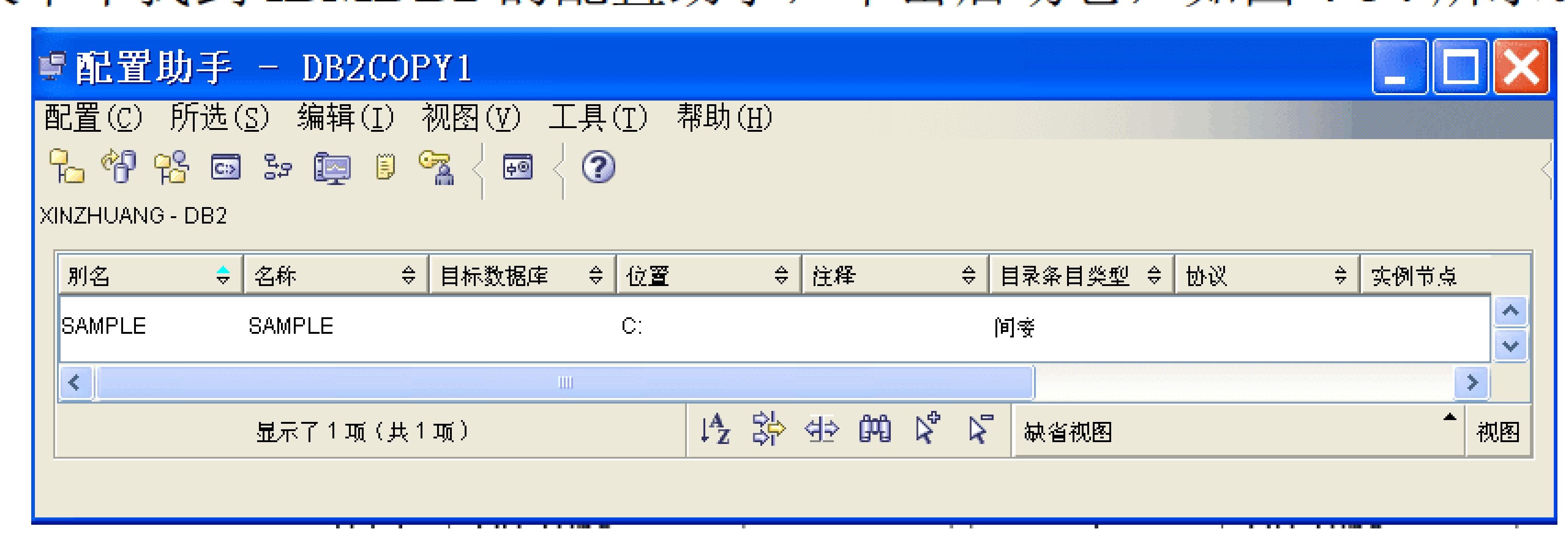


图 4-34 启动配置助手

(3) 打开“所选”菜单并选择“使用向导来添加数据库”子菜单。

(4) 在向导的“源”页面上，选择“手工配置与数据库的连接”选项，如图 4-35 所示。单击“下一步”按钮进入向导的下一页。

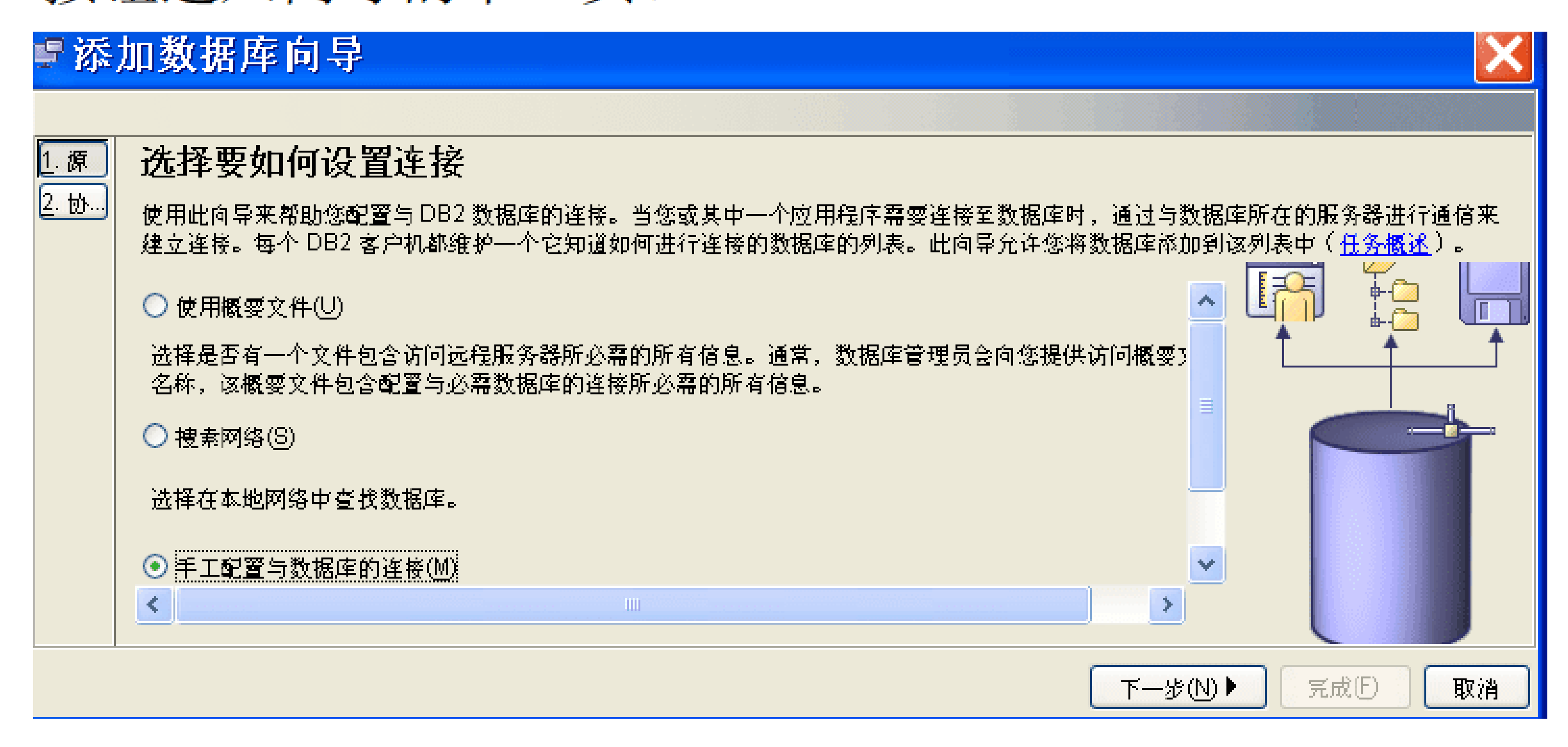


图 4-35 选择建立连接的方式

有 3 种连接方式：

- 使用概要文件：概要文件中包含访问远程服务器必需的所有信息(主机名、通信协议和端口)，适用于配置大批量客户端与服务器的通信。
- 搜索(Discover)网络：如果使用这种自动配置，那么不需要提供任何详细的通信信息，就能够让 DB2 客户机与 DB2 服务器进行联系。适用于本地网络，可以自动搜索到实例和数据库，但是需要在服务器端配置并启动数据库管理服务器(DAS)；在大型网络上，搜索可能要花费很长时间。
- 手工配置与数据库的连接：需要提供主机名、通信协议和通信端口。

(5) 在向导的“协议”页面上，选择 TCP/IP 选项，如图 4-36 所示。单击“下一步”按钮进入向导的下一页。

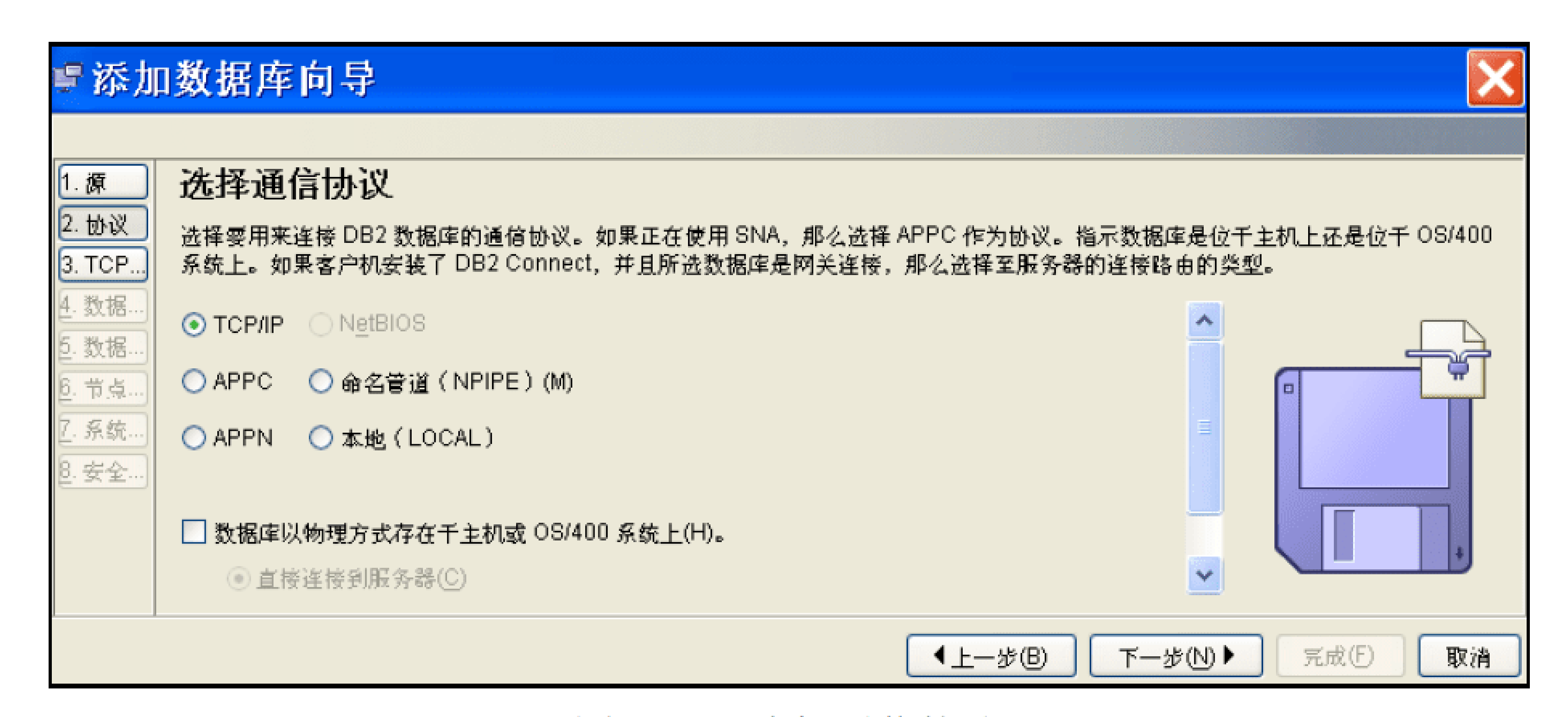


图 4-36　选择通信协议

(6) 在向导的 TCP/IP 页面上，输入在第(1)步中记下来的主机名或 IP 地址，以及端口号，如图 4-37 所示。单击“下一步”按钮进入向导的下一页。

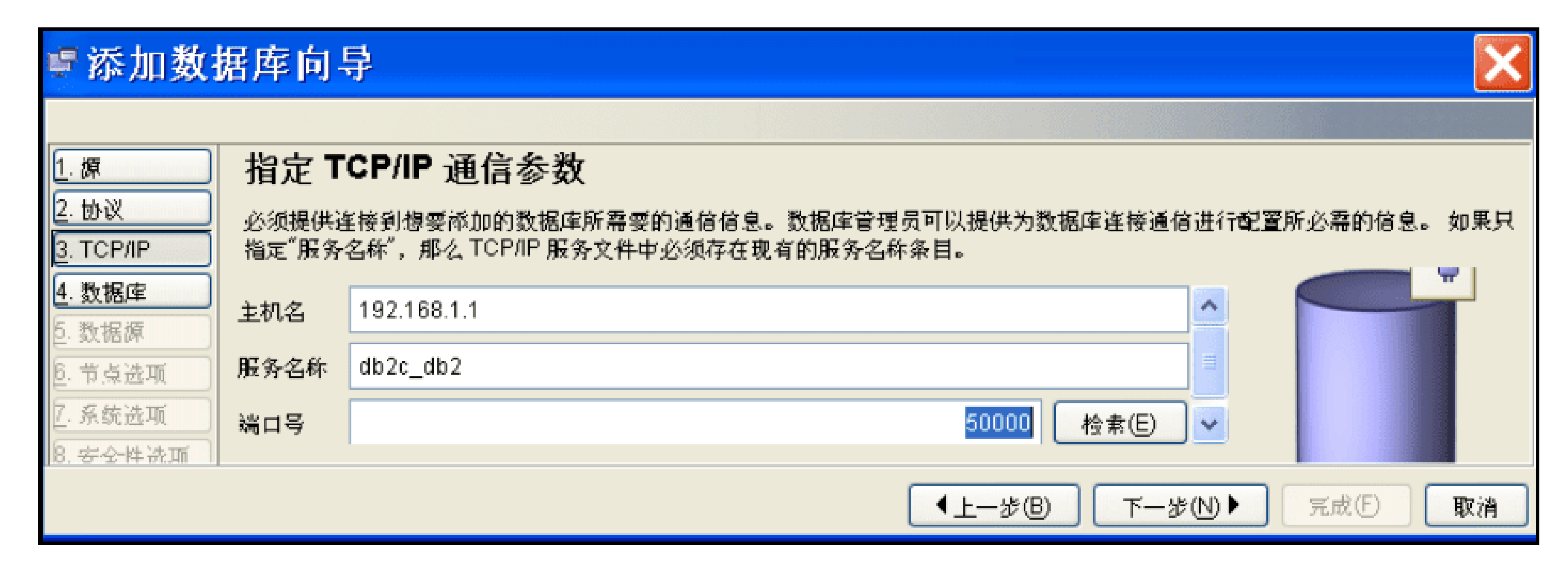


图 4-37　指定 TCP/IP 通信参数

注意：

如果在本地服务文件中有一个条目，其中定义了与远程服务器上实例监听的端口对应的端口号，那么可以使用“服务名称”(Service Name)选项。在使用这个选项时，DB2 会查看本地机器(而不是服务器)上的服务文件。如果要使用这个选项，就必须在这个文件中添加一个条目。

(7) 在向导的“数据库”页面的“数据库名称”框中，输入第(1)步中记下来的远程服务器上的数据库名。注意，“数据库别名(Database Alias)”框会自动填上相同的值，如图 4-38 所示。数据库别名是本地应用程序用来连接这个数据库的名称。因为您已经定义了一个名为 SAMPLE 的本地数据库，所以 DB2 不允许对同名的另一个数据库进行编目。因此，必须使用另一个别名。对于这个示例，将数据库别名改为 SAMPLE1。如果愿意，可以输入可选的注释。单击“下一步”按钮进入向导的下一页。

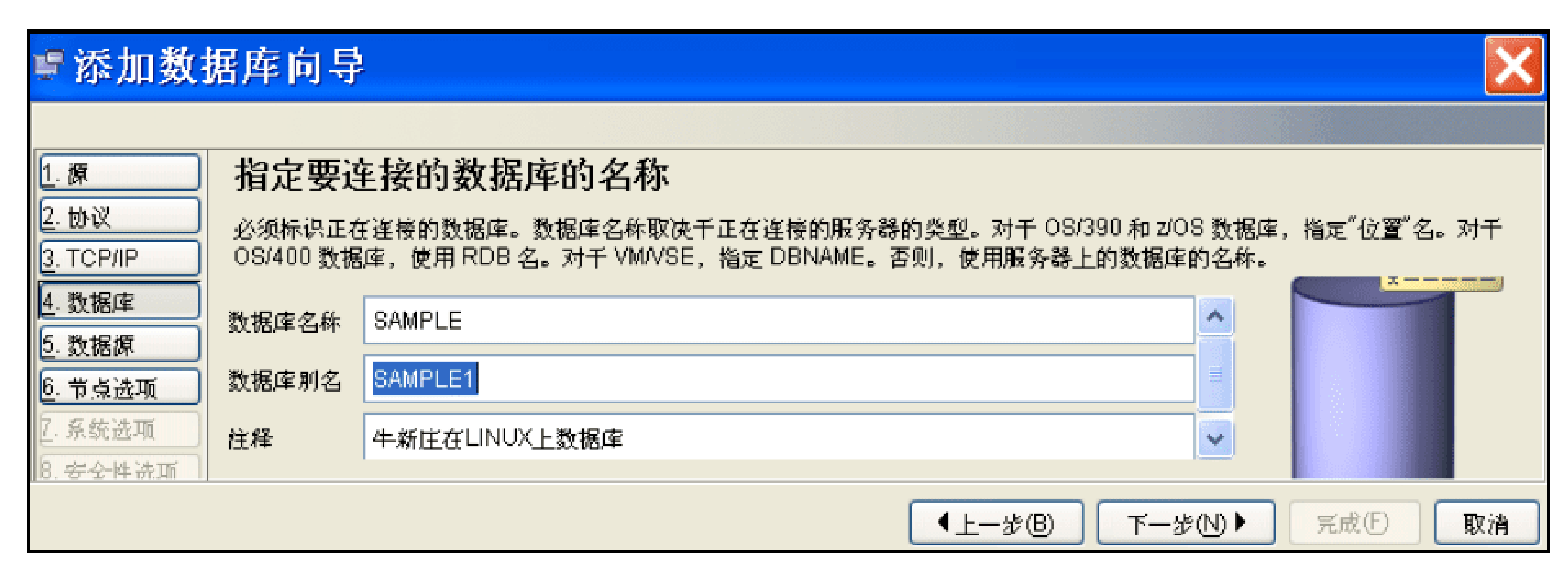


图 4-38　指定要连接的数据库的名称

(8) 在向导的“数据源(Data Source)”页面上，可以将这个新数据库(数据源)注册为 ODBC 数据源(这个步骤是可选的)。这会在 Windows ODBC Manager 中注册数据源。对于这个示例，因为不使用 ODBC，所以未选中“为 CLI/ODBC 注册此数据库”，如图 4-39 所示。单击 “下一步”按钮进入向导的下一页。

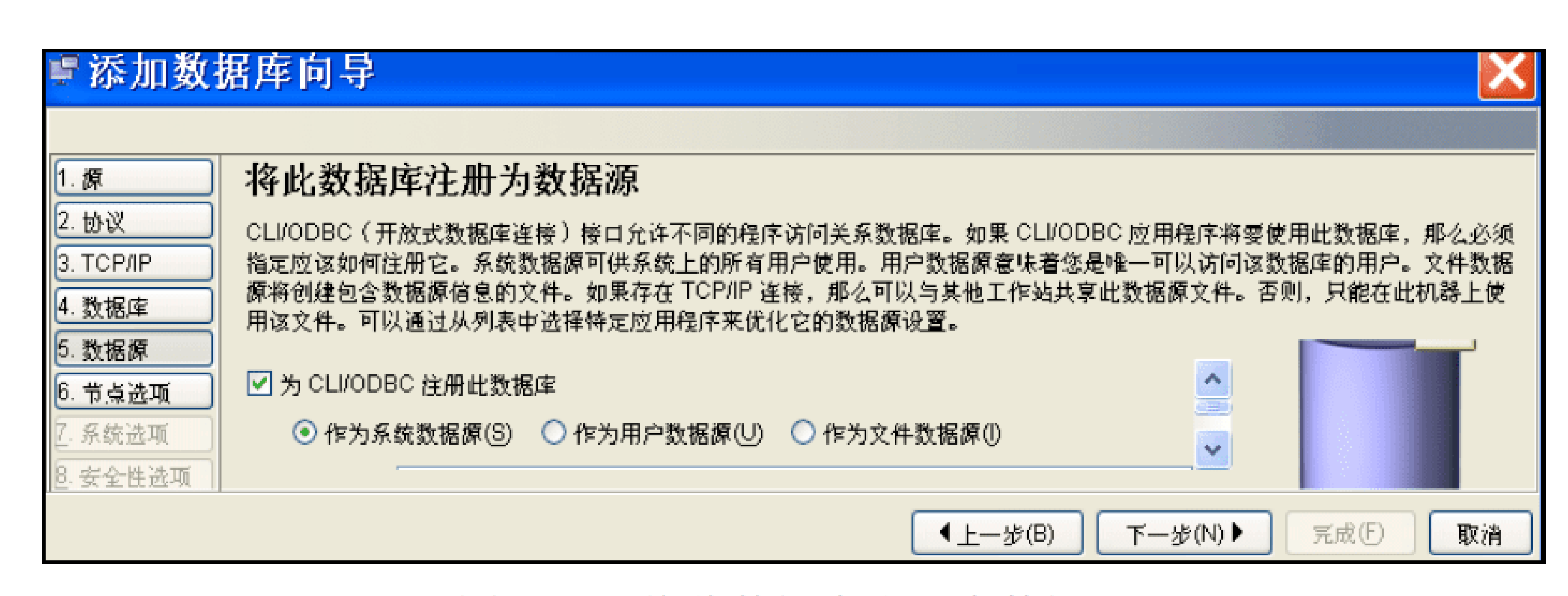


图 4-39　将此数据库注册为数据源

(9) 在向导的“节点选项(Node Options)”页面上，指定远程数据库所在的服务器的操作系统。这个实验中的所有工作站都使用 Microsoft Windows。确保在下拉列表中选择Windows。“实例名”框应该是 DB2。如果不是，就将它的值设置为 DB2，如图 4-40 所示。单击“下一步”按钮进入向导的下一页。

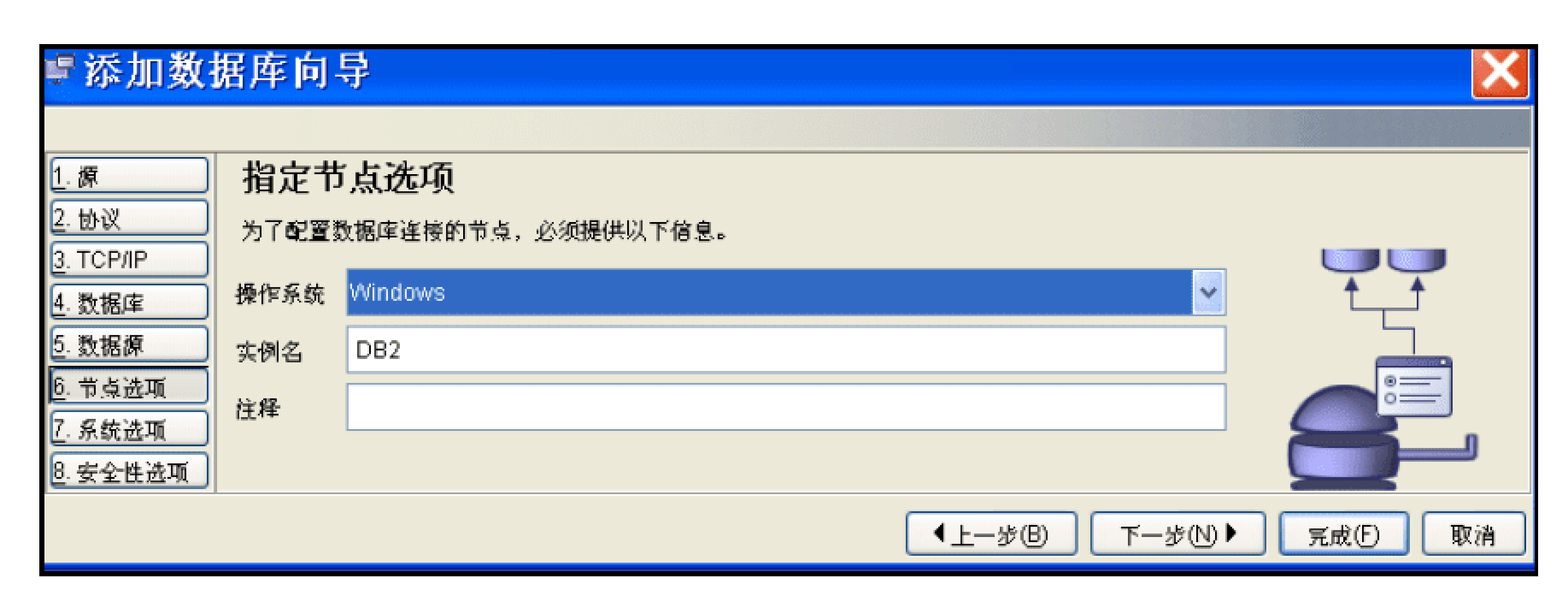


图 4-40　指定节点选项

(10) 在向导的“系统选项(System Options)”页面上，检查系统和主机名是否正确，检查操作系统设置，如图 4-41 所示。单击“下一步”按钮进入向导的下一页。



图 4-41　指定系统选项

(11) 在向导的“安全性选项”页面上，可以指定希望执行用户身份验证的位置和使用的身份验证方法，如图 4-42 所示。选择“使用服务器"DBM"中的认证值”选项，这将使用远程实例的配置文件中 AUTHENTICATION 参数指定的方法。单击“完成”按钮，对远程数据库进行编目并关闭向导。这时应该会出现一个确认框，如图 4-43 所示。单击“测试连接”按钮，确认可以成功地连接数据库。另外，这也会确认您提供的用户名和密码是远程服务器上定义的有效用户名和密码(因为服务器的 AUTHENTICATION 参数很可能设置为 SERVER)，如图 4-44 所示。如果连接测试成功，就会成功地对远程数据库进行编目。如果测试不成功，那么返回到向导中并检查指定的值是否正确(单击“更改”按钮可返回到向导设置)。如果您还想添加此远程服务器上的其他数据库，单击“添加”按钮继续添加数据库。

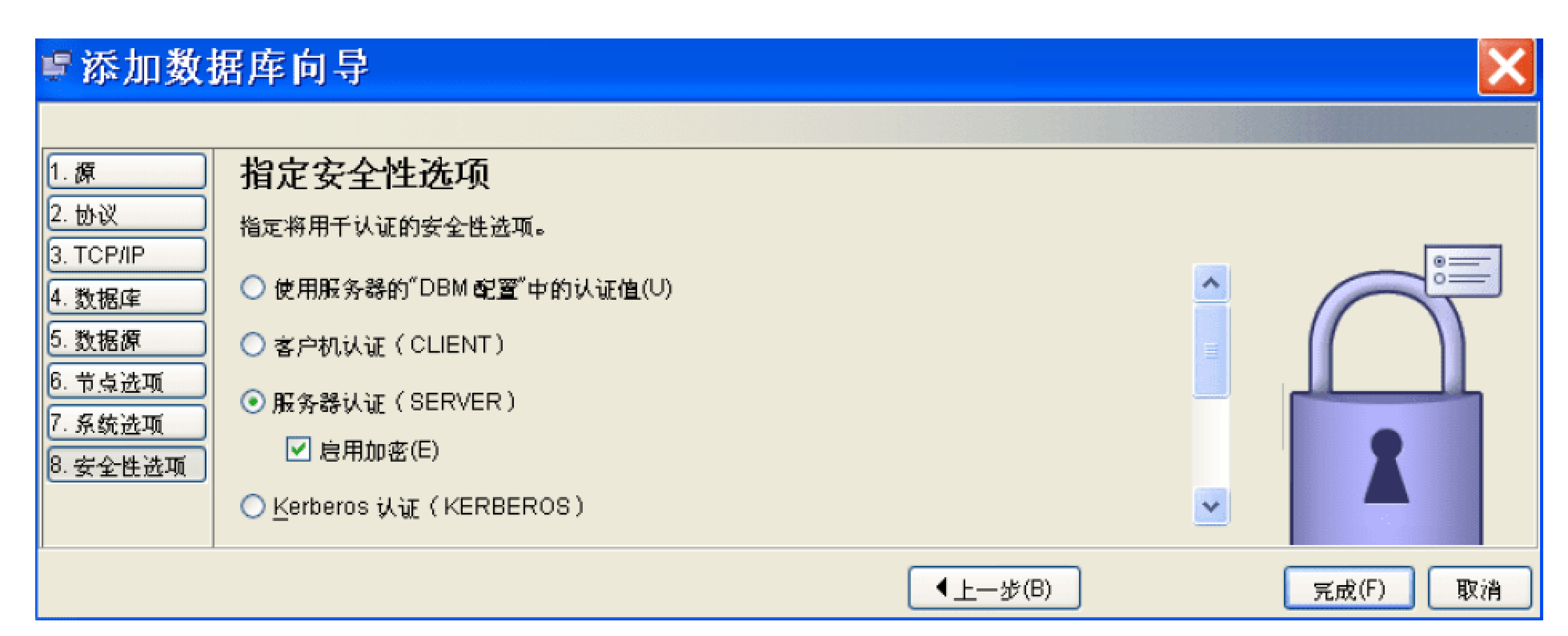


图 4-42　指定安全性选项

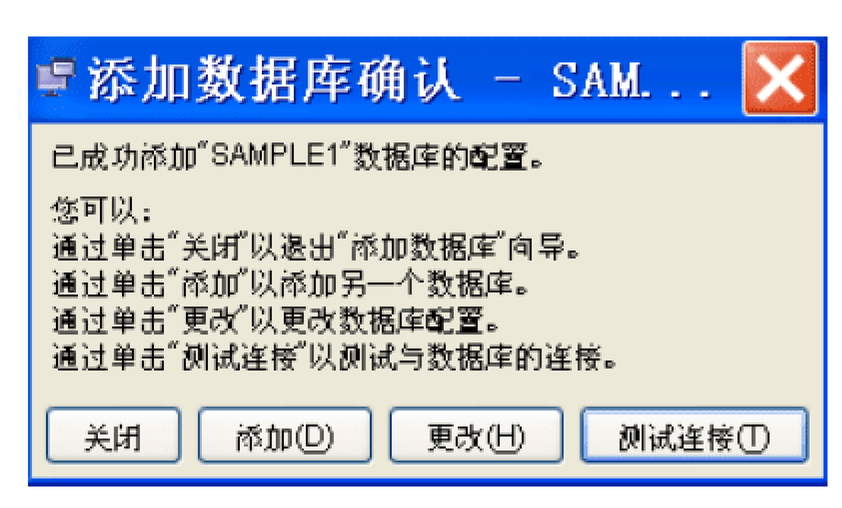


图 4-43　测试数据库连接

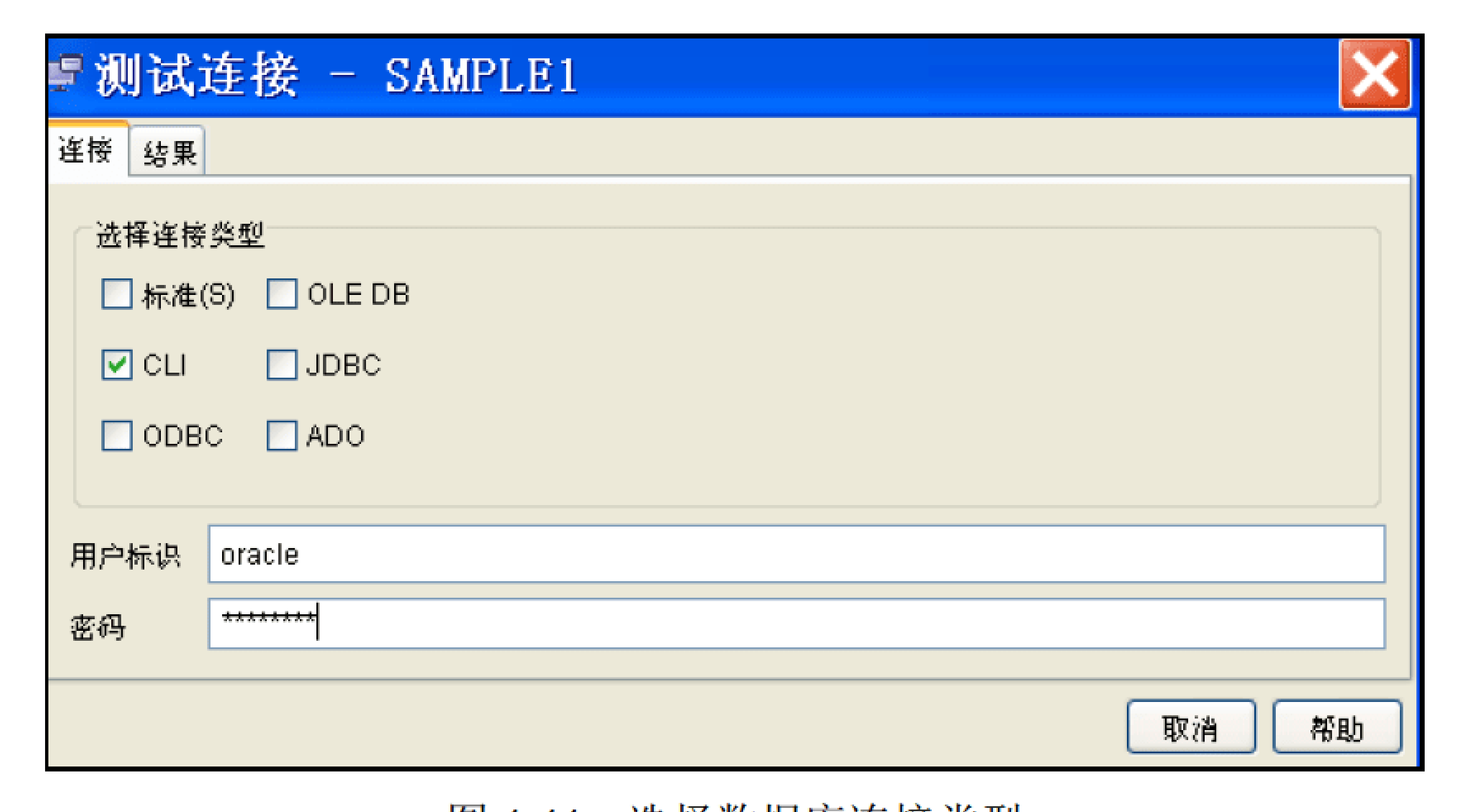


图 4-44　选择数据库连接类型

注意：

根据需要选择测试的连接类型，一般我们可以选择 JDBC，输入正确的“用户标识”和“密码”进行测试。

(12) 测试连接正确后，我们可以打开控制中心并尝试查看刚才编目的远程数据库以及其中的数据库表，如图 4-45 所示。



图 4-45　验证数据库编目是否正确

至此，我们已经使用配置助手配置好了客户机到服务器的通信。

4.5.4　深入了解 DB2 节点目录、数据库目录

在前面我们讲解了如何通过控制中心或 CA 来配置客户机到服务器通信，其实这两种方式本质上都是在后台调用命令来完成的。下面我们讲解如何使用命令行来配置客户机到服务器通信。在讲解客户机到服务器通信时，我们必须先弄清楚节点目录、系统数据库目录、本地数据库目录这几个概念。我记得我在刚开始学习 DB2 时，在成功地安装完之后，在我配置客户机通信时，我被文档上的“cataloging nodes and databases(编目节点和数据库)”这件事给弄糊涂了。catalog 这个词与过去惹人喜爱的 SYSCAT 和 SYSIBM 目录相比有着动词化的意味。有时候，我会对着 DB2 大声埋怨：“我不想编目任何东西，我只是想在远程客户端通过运行一个 SELECT 语句来确保我正确地安装了 DB2。”经过对节点目录和数据库目录概念的仔细研究，我了解到只有创建了一个数据库之后 DB2 才会有数据库目录；您不需要在本地机器上将节点和数据库编目——只有在一个连接到服务器的客户机上时才需要编目。

在 DB2 中，目录是存储有关系统、数据库及其连接信息的二进制文件。DB2 中有以下几种目录：

1. 节点目录

节点目录用于存储远程数据库的所有连通性信息。本节我们仅讲 TCP/IP 协议。在节点目录中大多数项将和 TCP/IP 信息有关，比如机器(其中包含了您想连接的数据库)的主机名或 IP 地址，还有相关的 DB2 实例的端口号。下面是一些与节点目录相关的命令：

- 要列示本地节点目录的内容，可使用 LIST NODE DIRECTORY 命令。请从 CLP 发出下面这个命令：

```
db2 list node directory
```

- 要将信息输入节点目录进行编目，请从 CLP 发出 catalog 命令：

```
db2  catalog TCPIP node <node name>  remote <hostname or IP address>
       server <port_name or port_number>
```

例如：

```
db2   catalog TCPIP node n1 remote 9.26.138.35    server 50000
```

- 要除去节点目录，请从 CLP 发出 uncatalog 命令：

```
db2  uncatalog node n1
```

要想得到您想要连接的远程实例的端口号，可以查看该实例的 dbm cfg 中的 svcename 参数来实现。该值通常对应于 TCP/IP services 文件中的某一项。

在每个数据库客户机上都创建并维护节点目录。对于具有客户机可以访问的一个或多个数据库的每个远程客户端，该目录都包含一个条目。无论何时请求数据库连接或实例连接，DB2 客户机都会使用该节点目录中的通信信息。该目录中的条目还包含客户机与远程实例通信要使用的通信协议的类型信息。在图 4-46 中，如果你想在 Workstation1 下的 inst1 实例下访问同一台机器上 inst2 实例和远程 Workstations2 上的 inst3 实例，那么必须在 inst1 下创建本地实例 inst2 和远程实例 inst3 的节点目录。

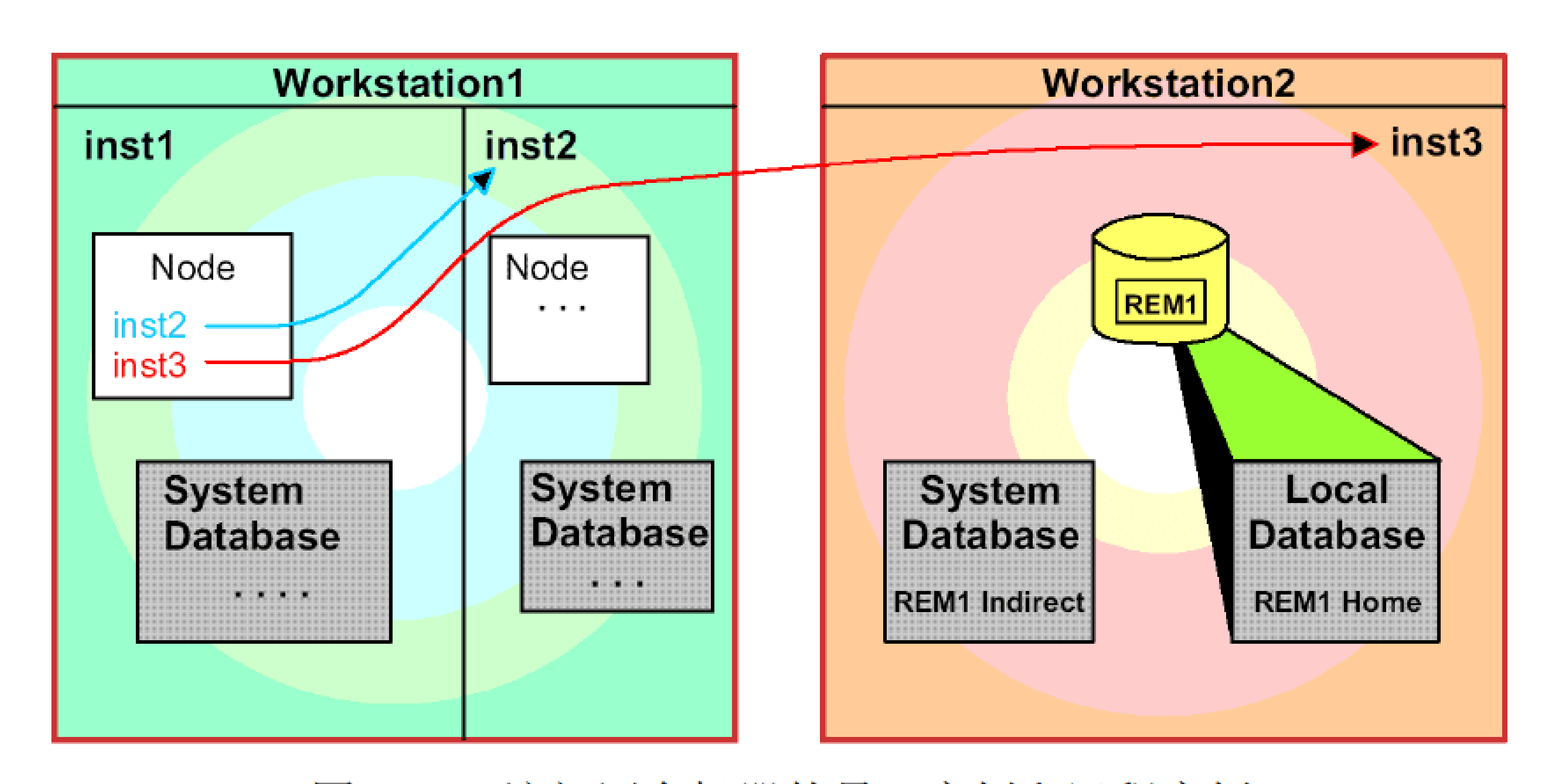


图 4-46　访问同台机器的另一实例和远程实例

可以这样理解，你要读取一个表，你必须先访问数据库；可是你要想访问数据库，那么必须先访问实例，因为数据库是包含在实例中的。但是我们无法直接访问实例，因为实例不是“物理”的，它是逻辑的，它是一组后端进程和共享内存的结合。所以这种情况下，我们就给实例建一个物理的“映射”，这就是节点目录的由来，所以节点目录是和实例对应的。如果实例就在本地，那么在创建实例的时候，默认会创建一个和实例同名的本地目录。这是隐式的，反正我们本地访问也用不到这个节点目录。但是如果在客户机上需要访问远程实例，那么必须为该实例建立一个和实例对应的节点目录。这个节点目录告诉我们该实例驻留在哪个机器上(IP 地址，主机名)，使用什么通信协议(设置 DB2COMM 变量)和使用的通信端口(SVCENAME)。

节点目录默认在实例目录下，有两个文件：SQLNODIR 和 SQLNOBAK。其中，SQLNOBAK 是 SQLNODIR 的备份，当 SQLNODIR 被损坏时，把 SQLNOBAK 改名为 SQLNODIR 即可。这个文件是二进制的，不过在 Windows 上您通过编辑器可以看到其中一些可读信息。

2. 系统数据库目录(或系统 db 目录)

系统数据库目录包含本地数据库目录和远程映射到本地的数据库目录。它是我们访问数据库的一个入口，我们连接数据库时首先去系统数据库目录中判断这个数据库是否存在，然后再判断这个数据库是本地数据库还是远程数据库。如果是本地数据库，就直接到本地物理目录上访问；如果是远程数据库，还要寻找这个远程数据库位于哪个节点上，然后再到节点目录中找到这个节点的通信信息。系统 db 目录是在实例级上进行存储的；对于数据库管理器的每个实例，都存在一个系统数据库目录文件，该文件对于针对此实例编目的每个数据库都包含一个条目。因此，如果您打算删除一个实例，那么您应当考虑备份其内容。

要列出系统 db 目录的内容，请从 CLP 发出下面这个命令：

```
db2 listdbdirectory
系统数据库目录
目录中的条目数 = 2
数据库 1 条目:
数据库别名                      = SAMPLE1
数据库名称                      = SAMPLE
节点名                          = BJ141
注释                            = 牛新庄在 LINUX 上数据库
目录条目类型                    = 远程
认证                            = SERVER_ENCRYPT
目录数据库分区号                = -1
备用服务器主机名                =
备用服务器端口号                =
```

```
数据库 2 条目:
数据库别名                    = SAMPLE11
数据库名称                    = SAMPLE
本地数据库目录                 = C:
数据库发行版级别               = c.00
注释                         =
目录条目类型                   = 间接(indirect)
目录数据库分区号               = 0
备用服务器主机名               =
备用服务器端口号               =
```

该命令输出中，任何包含单词“indirect”的项都意味着：该项适用于本地数据库(即，驻留在您当前正在使用的机器上的数据库)。该项还会指向由“Database drive”项(在 Windows 中)或“Local database directory”项(在 UNIX 中)所指示的本地数据库目录。

任何包含单词“Remote”的项都意味着：该项适用于远程数据库(即，驻留在其他机器上而非您当前正在使用的机器上的数据库)。该项还会指向由“Node name”项所指示的节点目录项。

要将信息输入系统 db 目录，您需要使用 catalog 命令：

```
db2 catalog db<db_name>  as <alias> at node  <nodename>
```

例如：

```
db2 catalog db  mydb  as yourdb  at node  mynode
```

节点名是指向节点目录中某一项的指针。在发出这条命令之前该项必须已经存在。

- 要除去数据库目录，请从 CLP 发出 uncatalog 命令：

```
db2  uncatalogdbsample1
```

通常只有在将信息添加到远程数据库的系统 db 目录时才使用 catalog 命令。对于本地数据库来说，当发出 CREATE DATABASE 命令创建数据库之后就自动创建 Catalog 项，将隐式地对数据库进行编目。

系统数据库目录中包含以下内容：

- 数据库名称、别名和注释
- 本地数据库目录的位置
- 目录条目类型(“remote”表示数据库在远程数据库；“indirect”表示是本地数据库)
- 节点名(此节点名和节点目录中的节点名匹配)

系统数据库目录默认在实例目录下，有 3 个文件：SQLDBDIR、SQLDBBAK 和

SQLDBINS，其中 SQLDBBAK 是 SQLDBDIR 的备份，当 SQLDBDIR 被损坏时，把 SQLDBBAK 改名为 SQLDBDIR 即可。文件 sqldbins 只有分区数据库才会用到，是指向共享文件系统中的另一个文件的符号链接。这些文件是二进制的，不过在 Windows 上您通过编辑器可以看到其中一些可读信息。

本地数据库目录(或本地 db 目录)

本地数据库目录包含了有关本地数据库(即，驻留在您目前正在使用的机器上的数据库)的信息。本地数据库目录驻留在数据库结构内部。当您用 create database 命令创建数据库时，将隐式地对数据库进行编目。在该目录中会添加一项。

要列出本地数据库目录的内容，请发出以下命令：

```
db2 listdbdirectory on c:
数据库 1 条目：

 数据库别名                          = BANK
 数据库名称                          = BANK
 数据库目录                          = SQL00002
 数据库发行版级别                    = c.00
注释                                =信贷 12 级分类数据库
 目录条目类型                        = 本地
 目录数据库分区号                    = 0
 数据库分区号                        = 0
```

其中，可以从系统 db 目录相应项中的“Database drive”项(Windows 中)或“Local database directory”项(UNIX 中)获取<path>。

节点目录，系统数据库目录和本地数据库目录之间的关系，如图 4-47 所示。

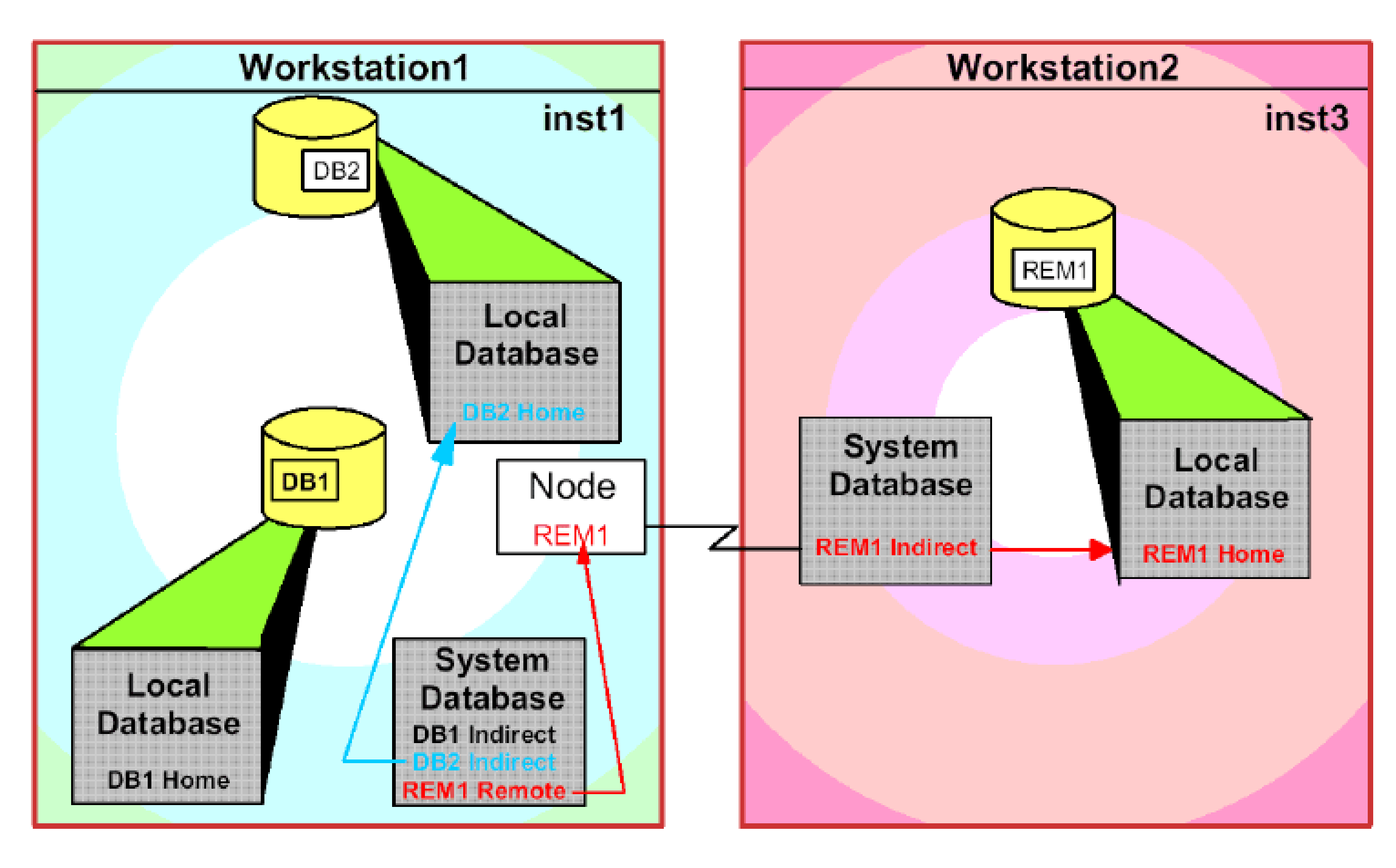
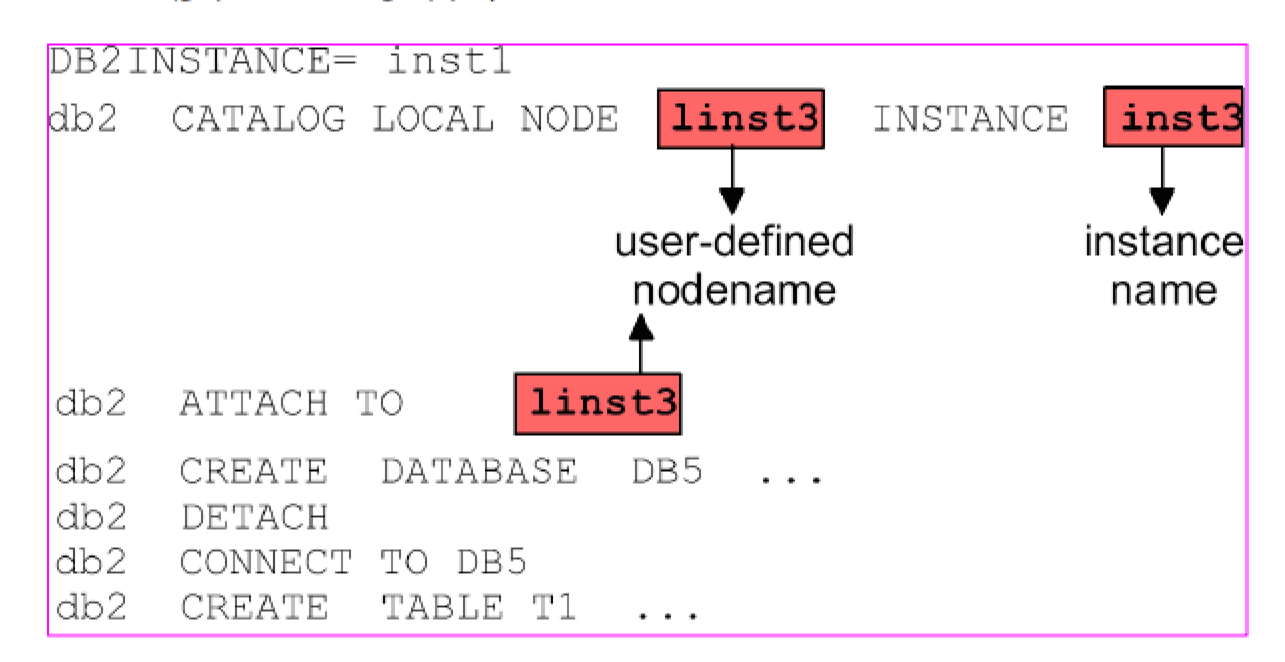


图 4-47　节点目录、系统数据库目录和本地数据库目录间的关系

如果在 workstation2 连接 inst3 实例：



如果要在 workstation1 上连接远程实例和远程数据库：

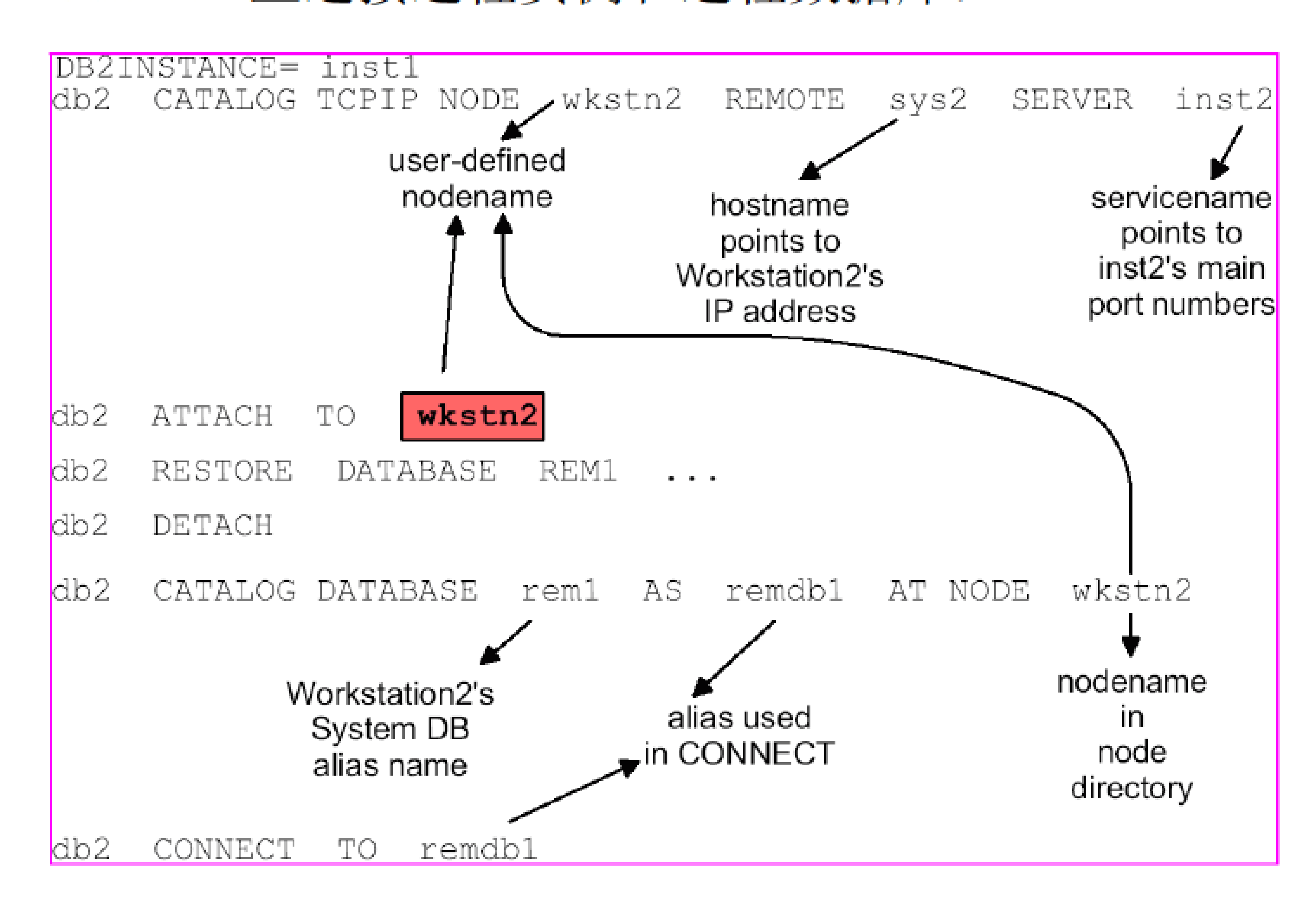


图 4-48 完整地总结了节点目录、系统数据库目录和本地数据库目录之间的关系。

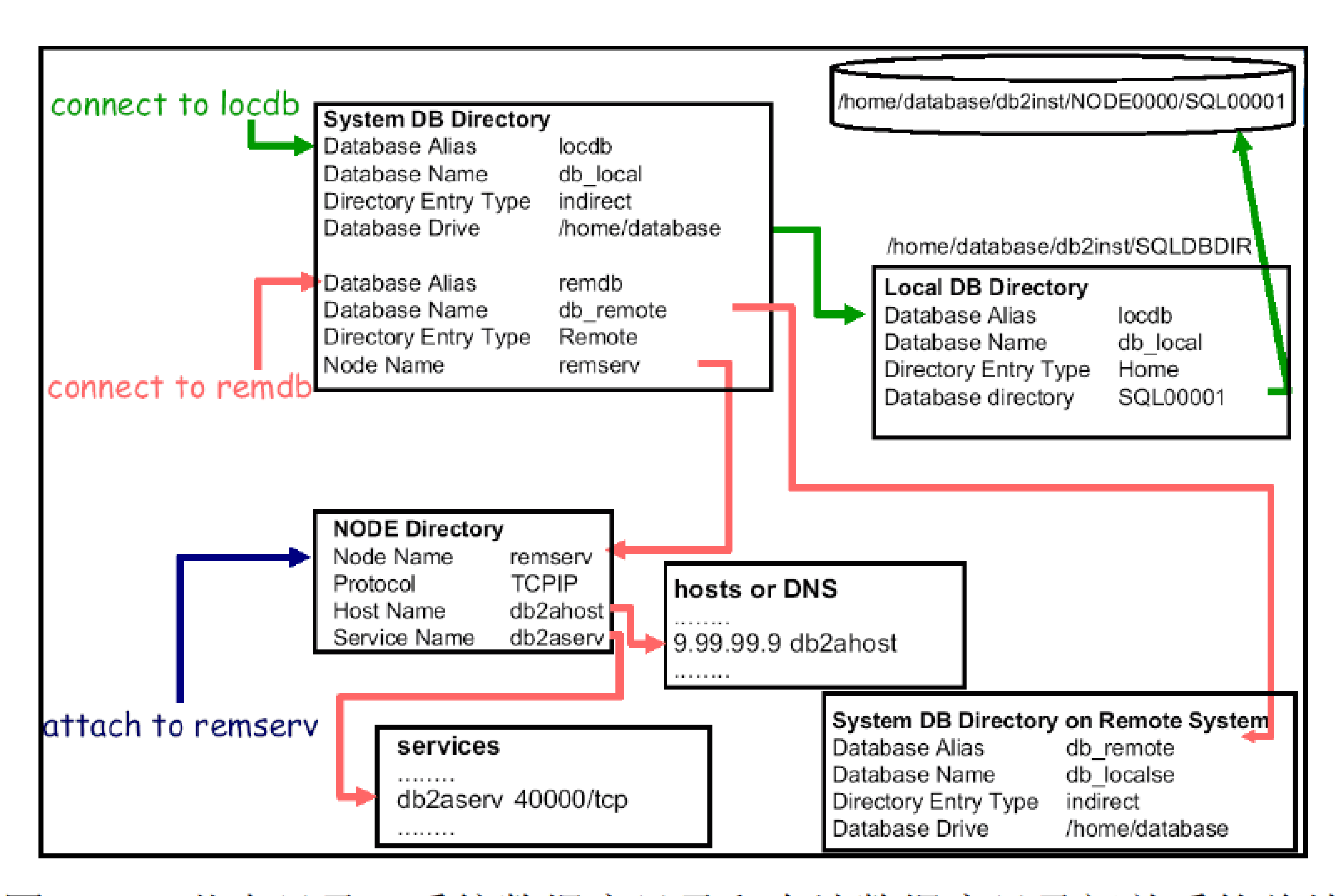


图 4-48　节点目录、系统数据库目录和本地数据库目录间关系的总结

4.5.5　使用 CLP 配置客户机到服务器通信案例

在了解了节点目录、系统数据库目录和本地数据库目录之后。如果我们想配置客户端到服务器端的通信，那么可以参考下面的图 4-49，它归纳了客户端到服务器端通信的所有配置参数，读者可以根据自己的实际情况填写这张表。

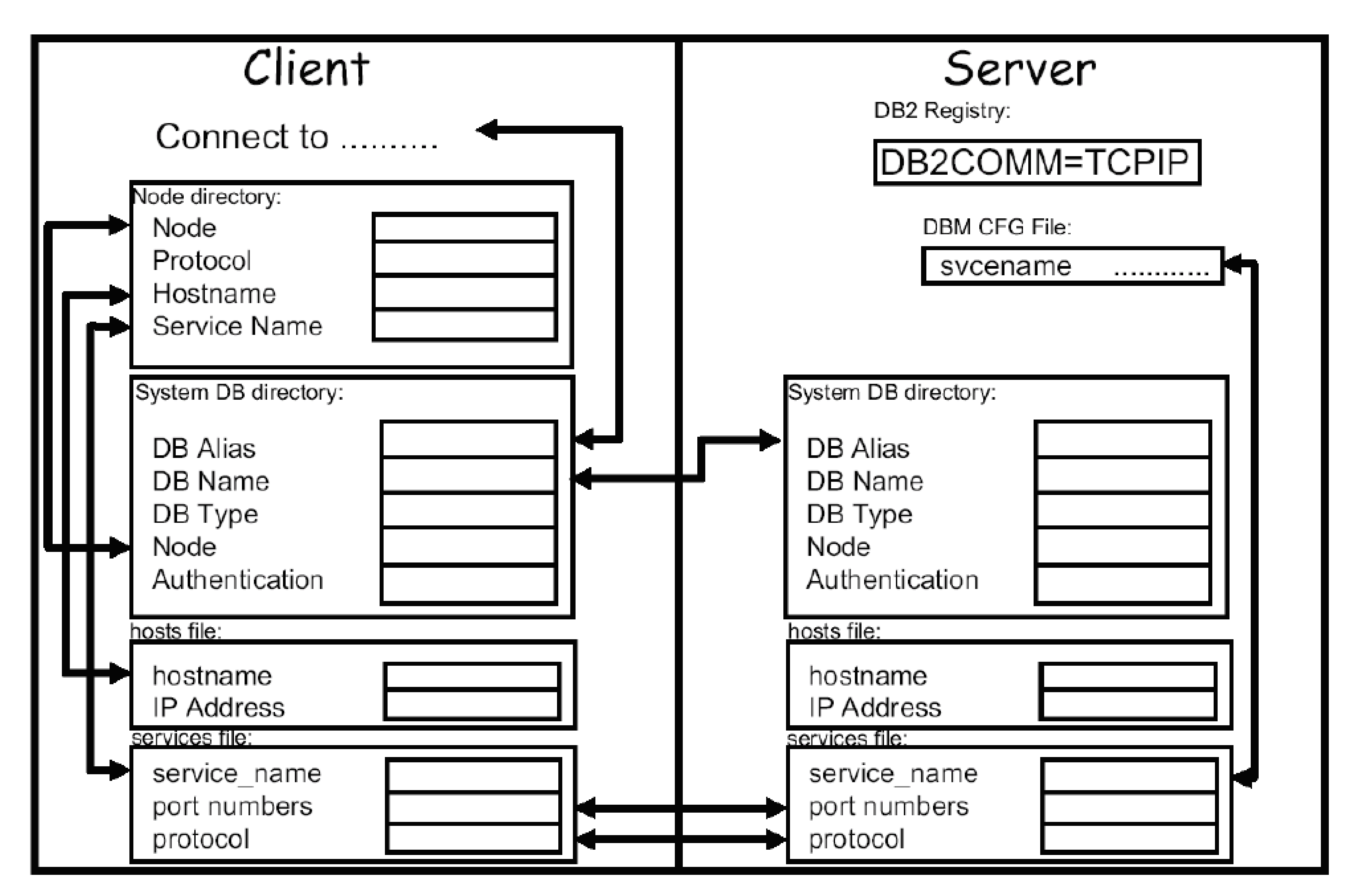


图 4-49　客户端到服务器端通信的配置参数

总的来说，要想配置客户端到服务器的通信，需要经过下面几个步骤；

(1) 更新 TCP/IP 连接的 hosts 和 services 文件。

如果要建立到远程数据库服务器的连接(通过使用它的主机名)，但是您的网络没有包含 DNS(域名服务器，用来解析主机名到 IP 地址)，那么必须更新 hosts 文件。如果要通过 IP 地址访问远程数据库服务器，那么不需要此步骤。如果要在建立与远程数据库服务器的连接时指定连接服务名称，那么需要更新 services 文件。连接服务是表示连接端口号的一个任意名称。如果要访问远程数据库服务器的端口号，那么不需要此步骤。

- 要更新客户机上的 hosts 文件以将远程服务器的主机名解析为它的 IP 地址：
  使用文本编辑器在 hosts 文件中添加一个条目，作为服务器的 IP 地址。例如：

```
11.21.15.235     myserver    # IP address for myserver
```

其中，11.21.15.235 表示 ip_address；myserver 表示 hostname

- 要更新客户机上的 services 文件以将服务名称解析为远程服务器的端口号：

使用文本编辑器将“连接服务名称”和端口号添加到 services 文件中。例如：

```
    server1  50000/tcp  #DB2connection service port
```

其中，server1 表示连接服务名称；50000 表示连接端口号(50000 为默认值)。

(2) 使用 CLP 从客户机编目 TCP/IP 节点。

编目 TCP/IP 节点会在数据服务器客户机节点目录中添加一个描述远程节点的条目。此条目指定客户机用来访问远程主机所选择的别名(node_name)、hostname(或 ip_address)和 svcename(或 port_number)。catalog 命令如下：

```
    db2 catalog TCPIP node node_name remote hostname|ip_address server
service name|port number [remote instance instance name]  [system system name]
[ostype  os_type]   -----[]内选项是可选的
    db2  terminate
```

要编目端口号为 50000、主机名为 server1、节点名为 db2node 的节点，应从 DB2 提示符处输入以下内容：

```
    db2  catalog TCPIP node db2node remote server1 server 50000
    DB20000I  CATALOG TCPIP NODE 命令成功完成。
    DB21056W  直到刷新目录高速缓存之后，目录更改才会生效。
    db2  terminate
    DB20000I  TERMINATE 命令成功完成。
```

(3) 使用 CLP 从客户机编目数据库。

必须先在客户机上编目数据库，客户机应用程序才能访问远程数据库。创建数据库时，除非指定了不同的数据库别名，否则将自动在服务器上以与数据库名称相同的数据库别名编目数据库。在数据服务器客户机上使用数据库目录中的信息和节点目录中的信息(除非要编目不需要节点的本地数据库)来建立与远程数据库的连接。在编目远程数据库时需要数据库名称、数据库别名、节点名、认证类型(可选)、注释(可选)等信息。catalog db 命令如下：

```
    db2 catalog database database_name as database_alias at   node node_name
[ authentication auth_value ]
```

要在使用认证 server1 的节点 *db2node* 上编目称为 *sample* 的远程数据库，以便它具有本地数据库别名 *mysample*，输入下列命令：

```
    db2  catalog database sample as mysample at node db2node   authentication server
    db2  terminate
```

(4) 用于编目数据库的参数表。

使用表 4-3 来记录编目数据库所需的参数值。

表 4-3　编目数据库所需的参数

参　　数	描　　述	样　本　值
数据库名称(database_name)	创建数据库时，除非另有指定，请将数据库别名设置为数据库名称。例如，在服务器上创建了 sample 数据库时，还将创建数据库别名 sample。数据库名称表示远程数据库别名(在服务器上)	sample
数据库别名(database_alias)	表示远程数据库的任意本地别名。若未提供别名，那么默认名称与数据库名称(database_name)相同。当从客户机连接至数据库时，使用此名称	mysample
认证(auth_value)	您的环境中所需的认证的类型	Server
节点名(node_name)	用来描述数据库存放位置的节点目录条目的名称。对用来编目节点的节点名(node_name)使用相同的值	db2node

(5) 使用 CLP 测试客户机至服务器连接。

在编目节点和数据库之后，应连接至数据库以测试连接。在测试连接之前：

- 数据库节点和数据库必须编目
- userid 和 password 的值对于认证它们所在的系统必须有效

要测试客户机与服务器的连接，在客户机上命令行输入以下命令以连接至远程数据库：

```
db2 => connect to database_alias user userid
```

例如：

```
db2 => connect to sample user informix using informix
```

如果连接成功，会接收到一条消息，显示已连接至的数据库的名称。将给出类似图 4-50 所示的消息。

现在就可以使用数据库了。例如，要检索系统目录表中列示的所有表名的列表，输入以下 SQL 语句：

```
db2 SELECT tabname from syscat.tables
```

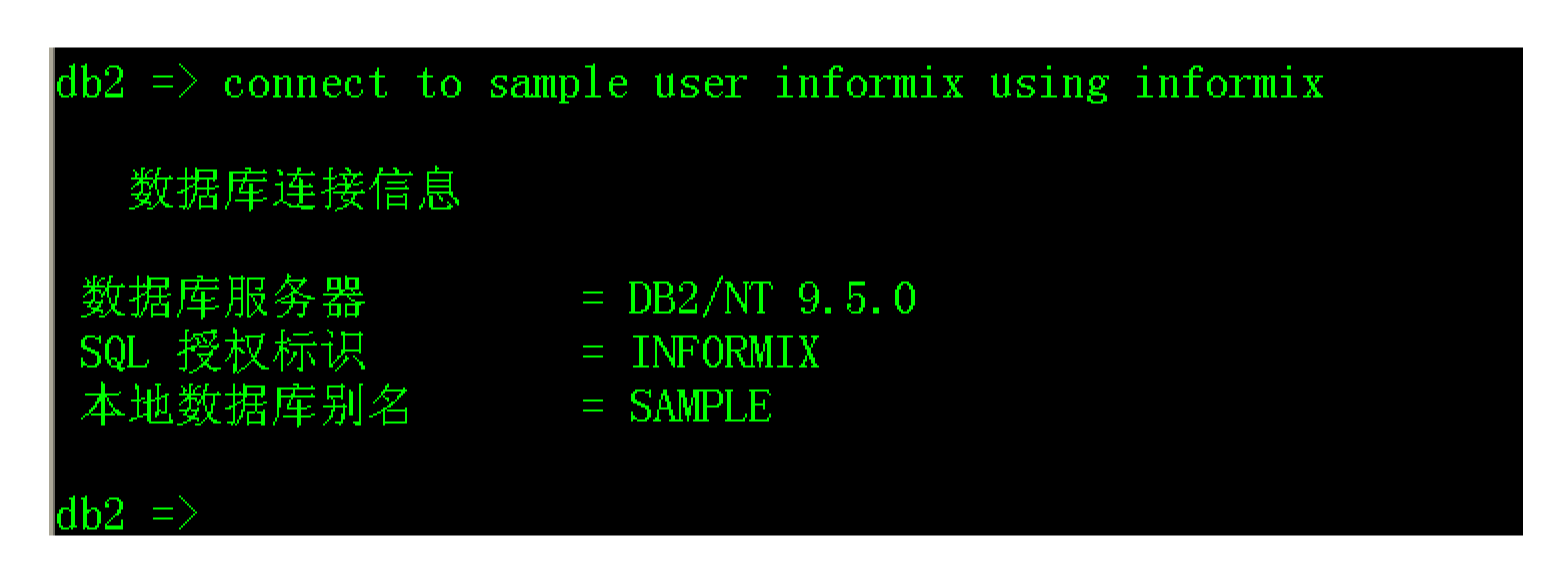


图 4-50 连接成功的消息

当结束使用数据库连接时，输入 connect reset 命令以结束该数据库连接。

(6) 配置客户机至服务器连接总结。

图 4-51 所示的这个检查列表总结了配置客服端到服务器端通信的主要检查项。

配置客户端连接服务器检查列表

- DB2SYSTEM注册变量已经设置为hostname，DB2安装期间默认已经设置为hostname，如果变更过hostname，请重新设置。
- 在服务器上设置DB2COMM注册变量
  db2set DB2COMM=tcpip
- 在services中设置实例端口，注意不要和别的端口冲突。
- 更新实例的配置文件，update dbm cfg using svcename db2c_db2
- 启动实例，用netstat命令检查端口状态是否为listening状态。
- 如果使用CA，需要启动DAS；如果使用CLP，在客户端catalog节点
  catalog tcpip node n1 remote hostname svcename 50000
- 在客户端catalog数据库，在我们已经编目的节点上
  catalog db sample at node n1
- 在客户端提供用户名和密码测试能否连接数据库；连接后如果能够读取数据说明配置连接成功。

图 4-51 检查列表

上述我们讲解了配置客户端到服务器端 TCP 通信的过程，其实在实际生产中，除了 TCP 通信，还有 APPC、APPN 和 NETBIOS 等很多通信协议。但是这些通信协议我们都不常用到，在这里就不讲解了。而且在某些环境中我们还会遇到一些其他组件：

- DB2 Connect 网关。这里指的是一个 DB2 Connect 服务器产品，它提供了一个网关，IBM 数据服务器客户机可通过该网关连接到中型机和大型机产品上的 DB2 服务器。

- LDAP(轻量级目录访问协议)。在一个启用了 LDAP 的环境中，不必配置客户机至服务器通信。当客户机试图连接至数据库时，若本地机器上的数据库目录中不存在该数据库，那么在 LDAP 目录中搜索连接数据库必需的信息。

当服务器设置为使用开发环境时(例如，IBM Data Studio)，您可能会在初始 DB2 连接时遇到错误消息 SQL30081N。可能的根本原因是远程数据库服务器的防火墙阻止建立连接。在这种情况下，请验证是否正确地配置了防火墙来接受客户机的连接请求。

4.6　本章小结

本章我们讲解了访问数据库的各种接口。重点讲解了如何配置客户端到服务器端的通信。一旦配置好访问数据库的接口，然后我们就可以访问数据库来创建数据库对象了。好，接着我们来讲解下一章：创建数据库对象。



创建数据库对象

在数据库创建后，我们可以根据我们的业务需求来设计和创建数据库对象了。可以在DB2 数据库中创建下列数据库对象：

- 模式
- 表
- 索引
- 序列
- 视图
- 触发器

我们可以使用图形用户界面或通过显式执行 SQL 语句来创建这些数据库对象。用于创建这些数据库对象的语句称为“数据定义语言(DDL)”，它们通常以关键字 CREATE 或 ALTER 作为前缀。

5.1 模式

5.1.1 模式概念

数据库中的大多数对象指定一个由两部分组成的唯一名称，如图 5-1 所示。第一部分(最左边的)称为限定词或模式，而第二部分(最右边的)称为简单(或未限定)名称。从句法上来说，这两部分并置成用句点分隔的单个字符串。第一次创建可以由模式名限定的任何对象(例如表、索引、视图、用户定义的数据类型、用户定义的函数、昵称、程序包或触发器)时，会根据对象名称中的限定词将该对象指定给一个特定模式。

Schema.OjbectName

图 5-1　数据库对象名的组成

例如，图 5-2 说明在创建表的过程中如何将该表指定给一个特定模式。

DB2 中的模式(schema)是一个已命名对象的集合，它提供一种方法来按逻辑分组这些对象。这些对象包括表、视图、索引、触发器、函数和包。模式提供了数据库中对象的逻辑类别。模式也是名称限定词；它提供一种方法来对几个对象使用相同名称，并防止对这些对象进行二义性引用。例如，使用模式名“PROD”和“DEV”很容易区分两个不同的 SALES 表(PROD.SALES 和 DEV.SALES)。模式名的最大长度为 30 字节，它用作分两部分的对象名的第一部分。例如，名称 CITIC.CUSTOMER。在这个示例中，CUSTOMER 表的完全限定名包含模式名：CITIC，这可以在系统编目中将它与其他名为 CUSTOMER 的表区分开。可以把模式想象为特定对象的创建者、生成者和主人。

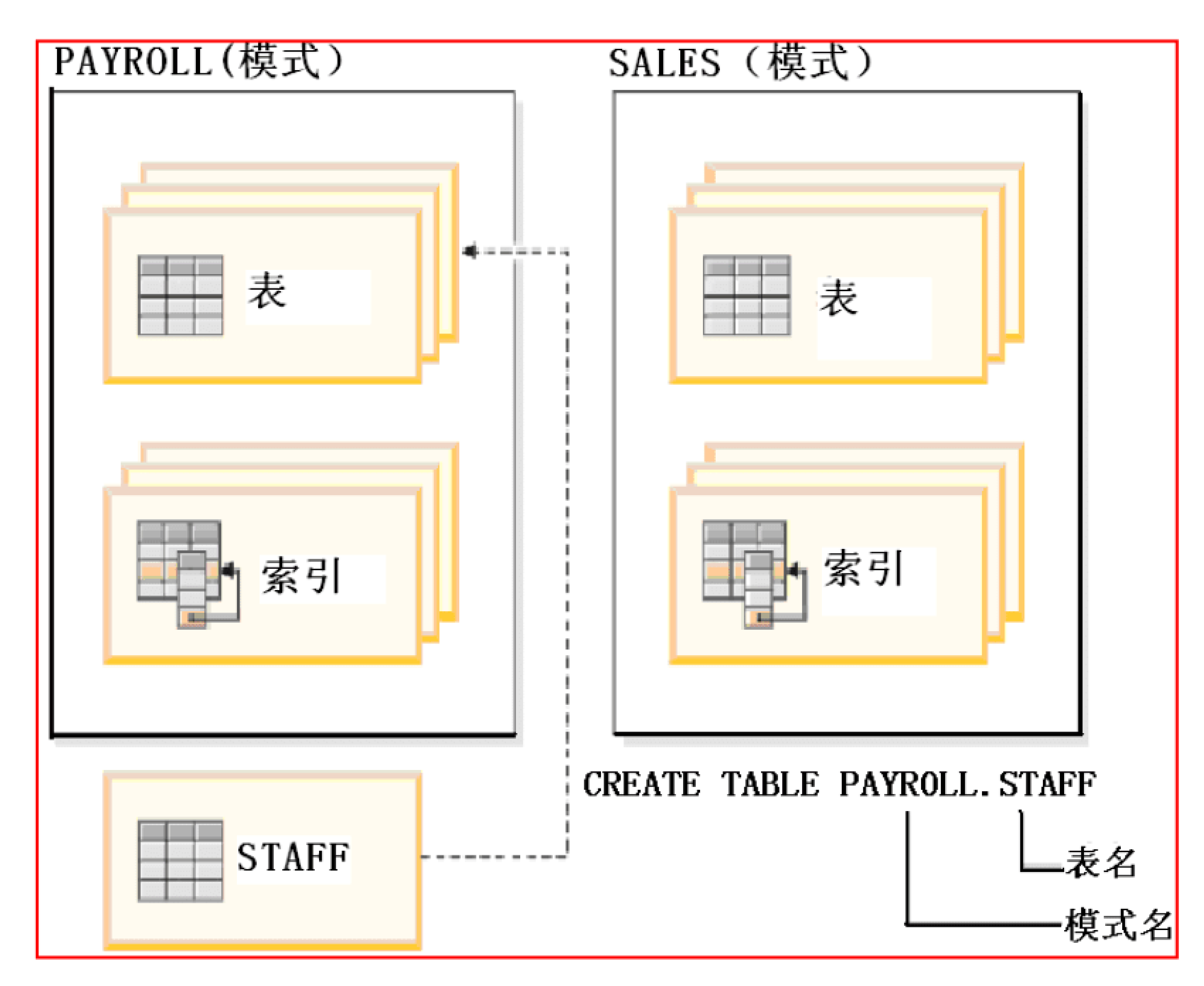


图 5-2　创建表并将其指定给一个特定格式

如果创建对象而没有指定模式，那么对象使用您的用户名与一个隐式模式相关联(假设用户或组具有 IMPLICIT_SCHEMA 数据库权限，IMPLICIT_SCHEMA 权限简单来说就是假设一个用户创建一个对象的时候没有使用模式，那么数据库就隐含地创建一个和用户名

一样的模式，关于这个权限在 13 章中有详细讲解)。当 SQL 语句引用对象时，如果没有指定模式名，那么也会隐式地加上调用者的用户名。

5.1.2　系统模式

对于每个数据库，都创建和维护一组系统编目表。这些表包含关于数据库对象(例如表、视图、索引和包)的定义的信息以及关于用户对这些对象的访问类型的安全信息。这些表存储在 SYSCATSPACE 表空间中，并采用保留的系统模式名：

- **SYSIBM、SYSFUN 和 SYSPROC：**一组例程，包括函数和存储过程，其中的 SYSIBM 是基本系统编目的模式(不建议直接访问它)。
- **SYSCAT：**一组只读的系统编目表视图，记录数据库对象的结构信息。
- **SYSSTAT：**一组可更新的编目视图。这些可更新的视图允许更新某些统计信息，从而模拟和测试数据库的性能，或者更新统计信息而不使用 RUNSTATS 实用程序。
- **SYSIBMADM：**一组动态性能视图，可以从该组视图中获取数据库的性能运行信息。在本书“第 9 章：DB2 性能监控”中有关于性能视图的详细讲解和案例。

5.1.3　设置和获得当前模式

在客户端连接实例或数据库时，会话的特殊寄存器 CURRENT SCHEMA 包含默认的限定符，用于对特定 DB2 连接中发出的动态 SQL 语句所引用的未限定对象进行限定。它的初始值等于特殊寄存器 USER 中的值(运行时用户)。静态 SQL 语句(在默认情况下)由绑定应用程序的用户的授权 ID 进行限定。用户可以使用 SET CURRENT SCHEMA 语句修改特殊寄存器 CURRENT SCHEMA 的值。

可以用 VALUES CURRENTSCHEMA 或 SELECT CURRENT SCHEMA FROM SYSIBM.SYSDUMMY1 命令获得当前的模式名。下面我们看两个使用模式的示例：

例 5-1　用户= HRUSER01，具有 IMPLICIT_SCHEMA 权限。

命令	结果
CREATE TABLE TEST1(ID INT, NAME VARCHAR(25))	Table HRUSER01.TEST1 created
CREATE TABLE CITIC.TEST1(ID INT, NAME VARCHAR(25))	Table CITIC.TEST1 created
SET CURRENT SCHEMA='CITIC'	CURRENT SCHEMA special register set to CITIC
INSERT INTO TEST1 VALUES(1,'John Doe')	Data inserted into table CITIC.TEST1

例 5-2 用户= HRUSER01，没有 IMPLICIT_SCHEMA 权限。

命令	结果
CREATE TABLE TEST1(ID INT, NAME VARCHAR(25))	SQL0552N "HRUSER01" does not have the privilege to perform operation "IMPLICIT CREATE SCHEMA". SQLSTATE=42502
CREATE TABLE HRUSER01.TEST1 (ID INT, NAME VARCHAR(25))	SQL0552N "HRUSER01" does not have the privilege to perform operation "IMPLICIT CREATE SCHEMA". SQLSTATE=42502
CREATE SCHEMA HRUSER01 AUTHORIZATION HRUSER01	Schema HRUSER01 created
CREATE TABLE TEST1(ID INT, NAME VARCHAR(25))	Table HRUSER01.TEST1 created

5.1.4 模式和用户的区别

我们要把模式和用户区分开，默认情况下一个用户(用户拥有 IMPLICIT_SCHEMA 权限)有一个和它同名的模式，您也可以根据需要创建模式授权给某个用户。模式创建有隐式创建和显式创建两种方式。

隐式创建

如果您具有 IMPLICIT_SCHEMA 权限，那么可以隐式创建模式。只要具有此权限，无论您何时使用不存在的模式名创建对象，都会隐式创建一个模式。只要创建对象的用户拥有 IMPLICIT_SCHEMA 权限，通常会在第一次创建模式中的数据对象时隐式创建模式。

显式创建

使用 CREATE SCHEMA 语句来创建模式。有关模式的信息保存在连接的数据库的系统目录表中。

要创建模式并让另一个用户成为该模式的所有者(后一个操作是可选的)，您需要 SYSADM 或 DBADM 权限。即使您不具有这两种权限中的任何一种，您也可以使用您自己的授权标识来创建模式。作为 CREATE SCHEMA 语句的一部分创建的任何对象的定义者是模式所有者。此所有者可以授予和撤销其他用户的模式特权。

要通过命令行来创建模式，请输入以下语句：

```
CREATE SCHEMA <schema-name> [ AUTHORIZATION <schema-owner-name> ]
```

其中<schema-name>是模式的名称。此名称在目录中已记录的模式内必须唯一，并且

不能以 SYS 开头。如果指定了可选的 AUTHORIZATION 子句，那么<schema-owner-name>将成为模式所有者。如果未指定此子句，那么发出此命令的授权标识将成为模式所有者。

例 5-3　下面的示例创建了 agent 模式并且把 agent 授权给 db2inst1 用户所有。

```
CREATE SCHEMA agent AUTHORIZATION db2inst1
```

删除模式

在删除模式之前，必须删除该模式中的所有对象或将它们移至另一个模式。当尝试 DROP 语句时，该模式名必须在语句中；否则会返回错误。

要使用命令行来删除模式，请输入：

```
DROP SCHEMA <name> RESTRICT
```

在以下示例中，删除了模式“agent”：

```
DROP SCHEMA agent RESTRICT
```

5.2　表设计考虑

所有数据都存储在数据库的表中。表由不同数据类型的一列或多列组成。数据存储在行(或称为记录)中。本节我不会过多地讲 CREATE TABLE、ALTER TABLE 或 DROP TABLE 之类的命令。这些命令您可以查 SQL 参考手册，本节主要讲一些和表设计相关的考虑事项，因为很多时候如果我们在建表的时候没有注意到这些，一旦系统上线，后期的调整往往非常麻烦，所以在建表之前，我们要作好规划设计。

5.2.1　选择合适的数据类型

定义列时，需要对列进行命名，定义这些列中将包含的数据的类型(称为数据类型)，并定义要创建的表中每列的数据长度。DB2 提供了一套丰富且灵活的数据类型。DB2 附带 INTEGER、CHAR、DATE 和大对象等基本数据类型。它还提供了创建用户定义的数据类型(UDT)的工具，使用户能够创建复杂的非传统的数据类型，从而适应当今复杂的编程环境。在给定的情况下，选用哪种数据类型取决于列中存储的信息的类型和范围。

内置的数据类型分为 5 类：数字、字符串、大对象、日期时间和 XML。

用户定义的数据类型分为：单值类型、结构化类型和引用类型(一般不用)。

DB2 内置的数据类型如图 5-3 所示。

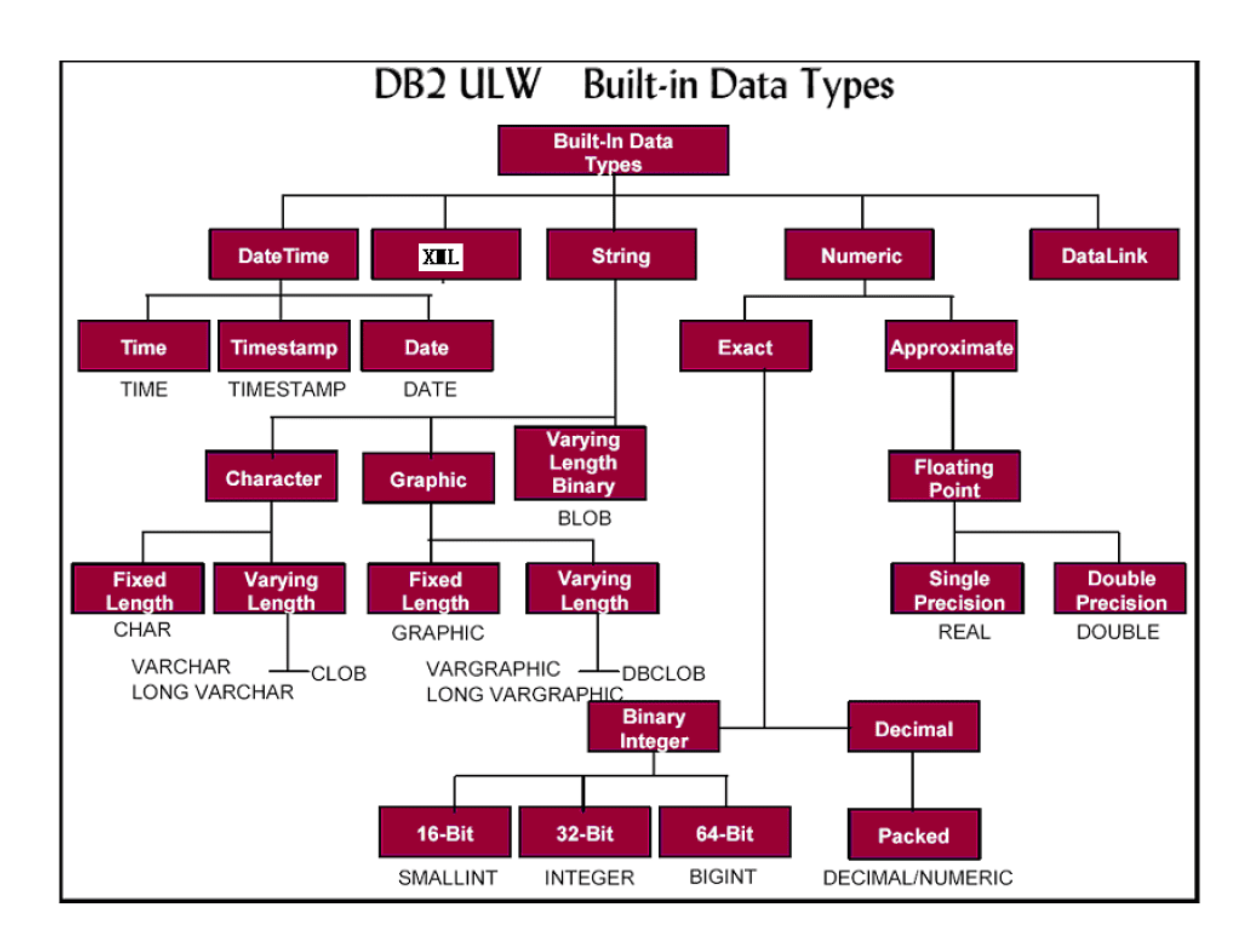


图 5-3　DB2 内置的数据类型

其中，XML 数据类型是 DB2 V9 以后版本提供的数据类型。DB2 提供了 XML 数据类型来存储格式良好的 XML 文档。XML 数据类型用于定义表中存储 XML 值的列，这些列中存储的所有 XML 值必须是结构良好的 XML 文档。引入此本机 XML 数据类型能够将结构良好的 XML 文档以其本机分层格式存储在数据库中其他关系数据旁边。

XML 列中的值存储为与字符串数据类型不同的内部表示。要在 XML 数据类型的列中存储 XML 数据，需要使用 XMLPARSE 函数对数据进行转换。可以使用 XMLSERIALIZE 函数将 XML 数据类型的值转换为 XML 文档的串行化字符串值。DB2 还提供了许多其他的内置函数来操纵 XML 数据类型。

我们在创建表时为列选择数据类型时一定要注意下面几点：

- 要根据业务需求选择合适的数据类型，避免出现数据类型转换。例如，我曾经看到有的客户使用字符来存放日期、时间戳，最后我们还要在程序中使用日期转换函数 to_date 作数据类型转换，这会对应用程序带来性能影响。
- 根据需求选择合适的长度。例如，用一个字段 empno 来存储员工号，用 SMALL INT 就可以满足，但是如果我们用 INT 就会造成两个字节的浪费。

- 如果某个字段的内容都是数字，建议大家选用整数而不要选用 CHAR。一个占用 4 字节的 INT 类型字段就可以表达到 4294967295，如果使用 CHAR 型则至少需要 10 个字节。一个占用 8 字节 LONG INT 类型字段就可以表达到 18446744073709551615，如果使用 CHAR 型至少需要 20 个字节。
- CHAR 和 VARCHAR 的选择，如果一列的数据有变化，但是变化不大时，而我们又追求性能，建议使用 CHAR 类型，因为 VARCHAR 的读取性能要分两个步骤，先读长度再读数据比 CHAR 的性能要弱些。
- LONG VARCHAR、BLOB、CLOB 和 CBLOB 数据类型的选择，这些大对象数据类型的读取是不经过内存而直接读取的，所以可根据情况看是否能够用 VARCHAR 字段代替。
- 如果使用大对象数据类型，考虑是否对该大对象列记录日志 NOT LOGGED。
- 考虑把大对象数据列单独存放在独立的表空间，和索引数据分隔存放。

下面让我们看看列在磁盘上是如何布局的。如果您创建了一个只有定长列的表，将严格按照 CREATE 语句中指定的顺序安排它们，如图 5-4 所示。

```
CREATE TABLE TESTORD(COL1 INT, COL2 CHAR(5), COL3 DEC(10,2), COL4 FLOAT)
```

col1 int	col2 varchar(5)	col3 dec(10,2)	col4 float

图 5-4　定长列的表在磁盘上布局

如果表拥有变长列(如 VARCHAR)，列仍然按照 CREATE TABLE 语句中指定的顺序排序，但可变数据本身在行的末尾，如图 5-5 所示。

```
CREATE TABLE TESTORD(COL1 INT, COL2 VARCHAR(5), COL3 DEC(10,2))
```

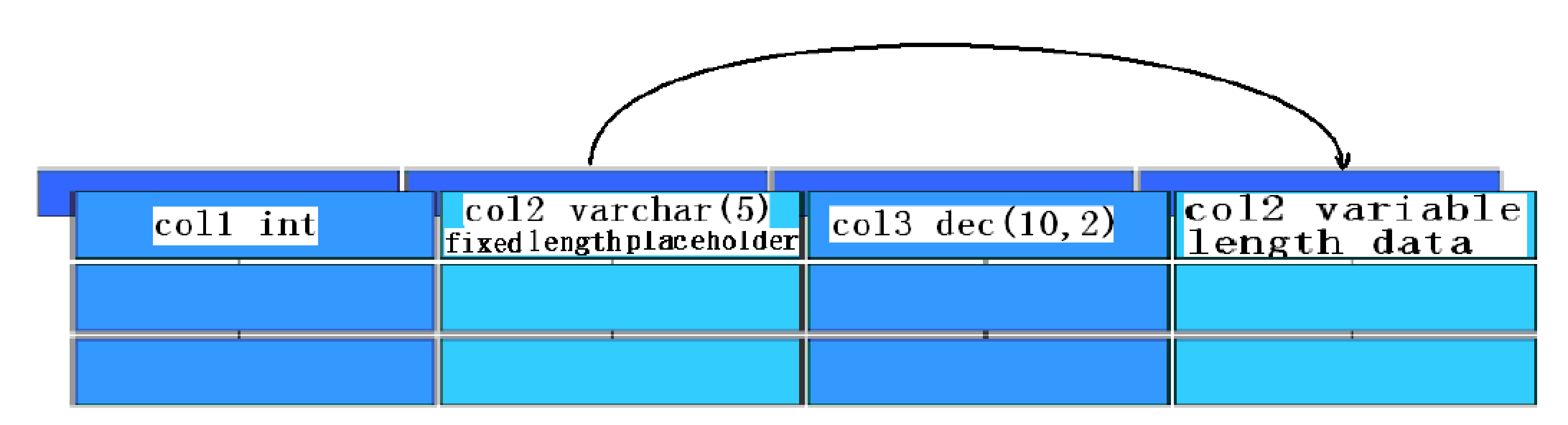


图 5-5　拥有变长列的表在磁盘上的布局

如果表有长字段，它将不随每行直接插入。因为行的长度受页大小限制(4KB 到32KB)，所以行只有一个指向长字段的指针，而将长字段与行分开放置在数据库页中，如图 5-6 所示。

```
CREATE TABLE TESTORD(COL1 INT, COL2 CLOB(100 K), COL3 DEC(10,2))
```

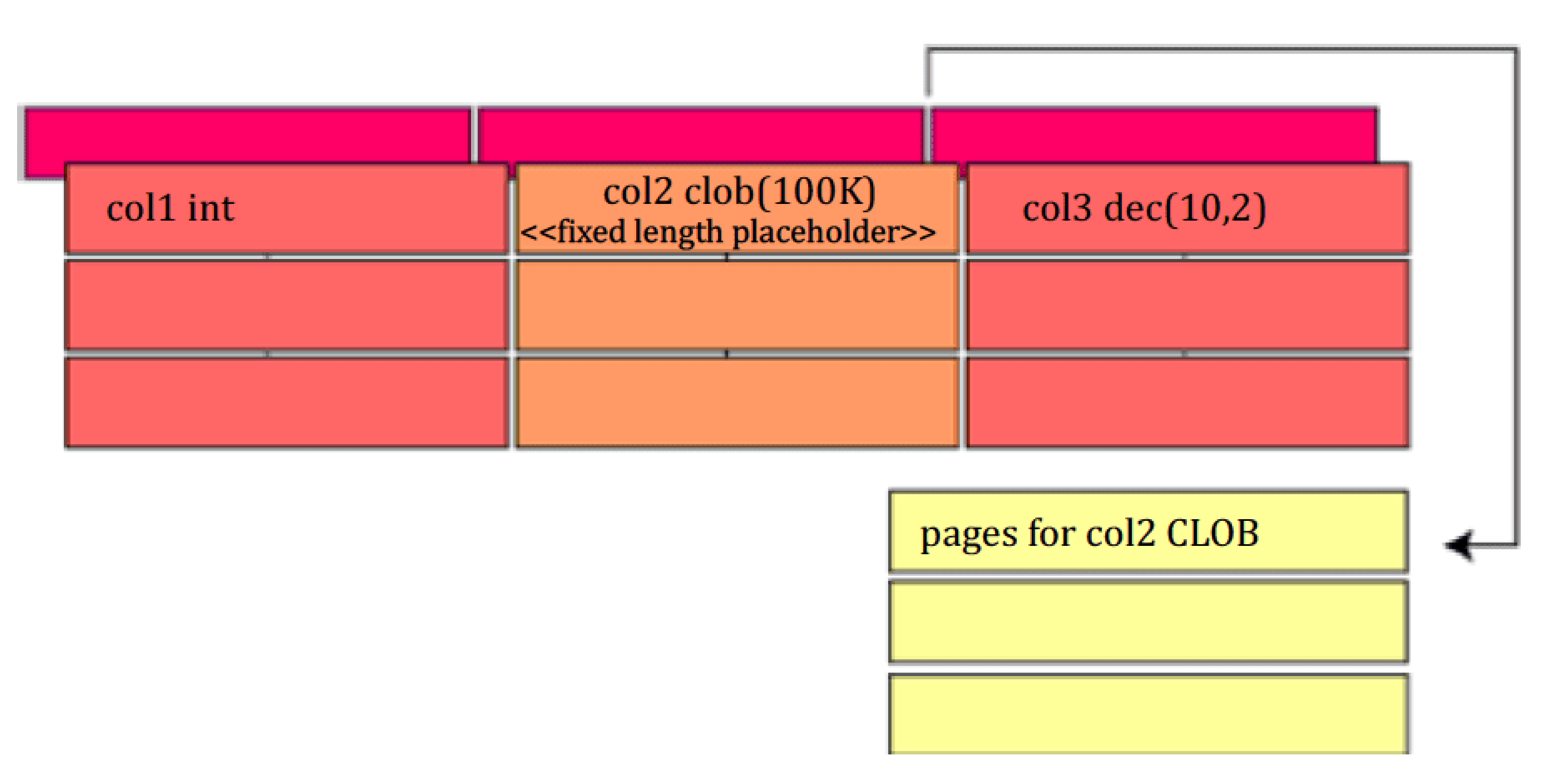


图 5-6　拥有长字段的表在磁盘上的布局

5.2.2　选择合适的约束类型

在任何业务中，数据通常必须符合特定限制或业务规则。例如，职员编号、银行支票号必须是唯一的。数据库管理器提供了约束作为强制实施这种规则的方法。约束是用于业务需求的规则。DB2 提供了下列 5 种类型的约束：

NOT NULL 约束

NOT NULL 约束防止在列中输入空值。NOT NULL 约束是这样一种规则，它防止在表的一列或多列中输入空值。数据库中使用空值来表示未知状态。默认情况下，随数据库管理器一起提供的所有内置数据类型都支持空值的存在。但是，一些业务规则可能要求必须始终提供值(例如，乘飞机时必须提供紧急联系人信息)。NOT NULL 约束用于确保决不会为给定表列指定空值。为特定列定义 NOT NULL 约束后，尝试在该列中放入空值的任何插入或更新操作将失败。

唯一约束

唯一约束确保一组列中的值对于表中的所有行都是唯一的，且不为空。在唯一约束中指定的列必须定义为 NOT NULL。唯一约束(也称为唯一键约束)是这样一种规则，它禁止

表的一列或多列中出现重复值。唯一键和主键是受支持的唯一约束。例如，可对供应商表中的供应商标识定义唯一约束以确保不会对两个供应商指定同一供应商标识。唯一约束确保一组列中的值对于表中的所有行都是唯一的，且不为空。在唯一约束中指定的列必须定义为 NOT NULL。数据库管理器使用唯一索引在对唯一约束的各列进行更改时强制键的唯一性。例如，DEPARTMENT 表中的典型唯一约束可以是：部门号是唯一的，且不为空。

图 5-7 显示了当表存在唯一约束时，阻止将重复的记录添加到该表。

数据库管理器在插入和更新操作期间强制执行此约束，以确保数据完整性。表可以有任意数目的唯一约束，但最多将一个唯一约束定义为主键。对于同一组列，表不能有多个唯一约束。

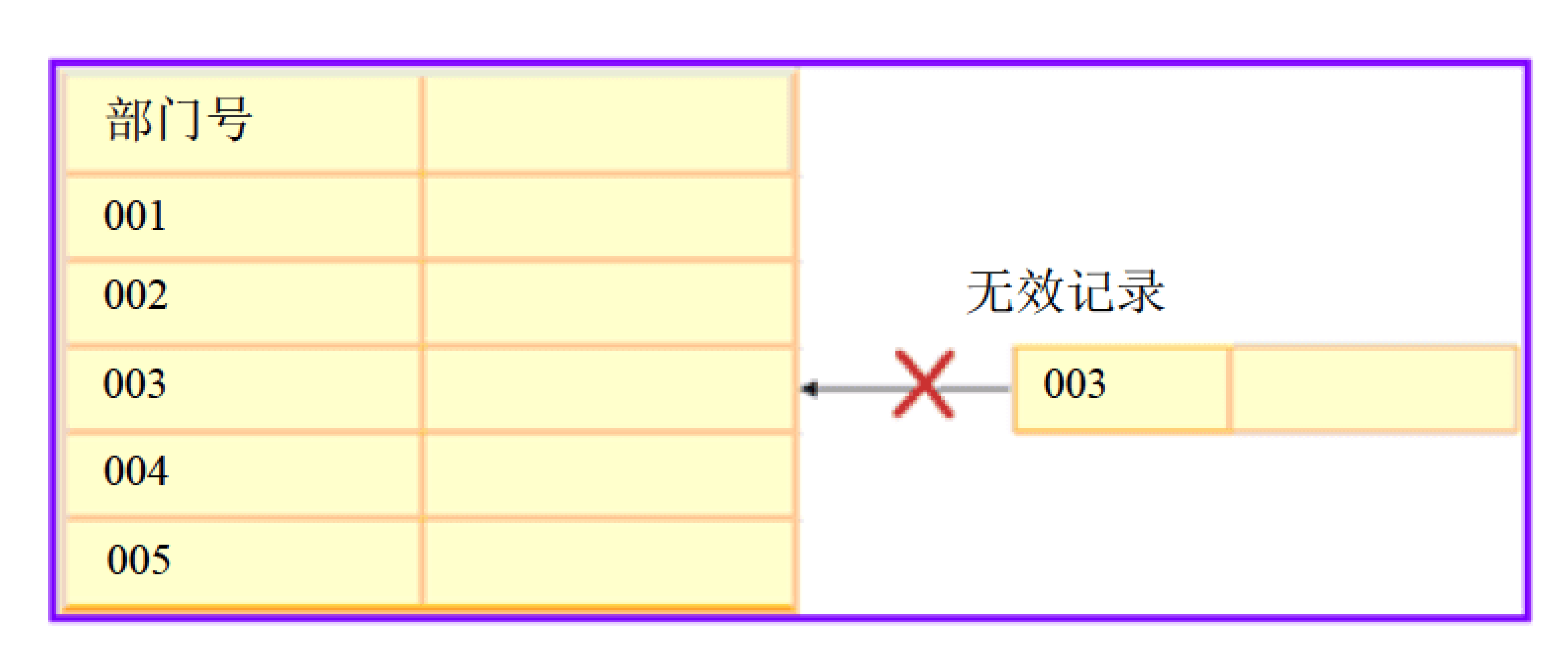


图 5-7　唯一约束防止出现重复数据

- 当在 CREATE TABLE 语句中定义唯一约束时，唯一索引是由数据库管理器自动创建的，且被指定为主索引或系统所需的唯一索引。
- 当在 ALTER TABLE 语句中定义唯一约束且同一组列存在索引时，该索引被指定为唯一的且是系统所需的。如果这样的索引不存在，数据库管理器会自动创建唯一索引，并将其指定为主索引或系统所需的唯一索引。

注意：

定义唯一约束与创建唯一索引是有区别的。尽管都强制唯一性，但唯一索引允许可空列，且通常不能用作参考约束的父键。

主键约束

主键是与唯一约束具有相同属性的一个列或列的组合。因为主键用来标识表中的一行，所以它必须是唯一的，并且必须具有 NOT NULL 属性。一个表不能有多个主键，但可以有多个唯一键。主键是可选的，可以在创建或改变表时定义。当导出或重组数据时，主键可以对数据进行排序，所以它们也是有益的。

(表)检查约束

检查约束(也称为表检查约束)是这样一种数据库规则，它指定表中每行的一列或多列中允许使用的值。指定检查约束是通过限制格式的搜索条件完成的。*检查约束*对添加至特定表的数据设置限制。例如，表检查约束可确保每当在包含个人信息的表中添加或更新薪水数据时，职员的薪水级别至少为￥2000。

外键(参考)约束

外键约束(也称为参考约束或参考完整性约束)使您能够定义表间以及表内必需的关系。外键约束是关于一个或多个表中的一列或多列中的值的一种逻辑规则。例如，一组表共享关于公司的供应商的信息。供应商的名称有时可能会更改。可定义一个参考约束，声明表中的供应商的标识必须与供应商信息中的供应商标识相匹配。此约束会阻止外键约束，使您能够定义表间以及表内必需的关系。

例如，典型的外键约束可能规定 EMPLOYEE 表中的每个职员必须是一个现有部门的成员，该部门在 DEPARTMENT 表中定义。

参考完整性是数据库的一种状态，在该状态中，外键的所有值都有效。外键是表中的一列或一组列，它的值需要与其父表的行的至少一个主键或唯一键值相匹配。参考约束是这样一种规则，仅当满足下列其中一个条件时，外键的值才有效：

- 它们作为父键的值出现
- 为空

要建立此关系，应将 EMPLOYEE 表中的部门号定义成外键，并将 DEPARTMENT 表中的部门号定义成主键。图 5-8 显示了当两个表之间存在外键约束时，如何阻止将具有无效键的记录添加至表。



图 5-8　外键和主键约束

包含父键的表称为参考约束的父表，包含外键的表被认为是该表的从属表。

在了解了以上 5 种类型的约束后，我们在进行表的设计时，可以根据自己的业务需要来决定创建什么类型的约束。

5.2.3　使用 not null with default

在数据库中，NULL 值是最难处理的，我们编程的时候对 NULL 值的处理也颇费周折，例如在嵌入 SQL 编程中，为了处理 NULL 值，我们必须在程序中声明一个指示符变量，这增加了编程的成本。所以我的建议创建表时对列使用 not null with default 选项，这样如果该列为 NULL，就使用默认值代替 NULL。

例 5-3　使用 not null with default 创建。

```
C:\>db2 create table xinzhuang tab(id int NOT NULL with default 1)
DB20000I  SQL 命令成功完成。
```

5.2.4　生成列及应用案例

生成列在表中定义，在这些列中，存储的值是使用表达式计算得出的，而不是通过插入或更新操作指定。

当创建已知始终要使用特定表达式或谓词的表时，可对该表添加一个或多个生成列。通过使用生成列，就有机会在查询表数据时改进性能。

例如，当性能很重要时，以下两种表达式求值方式成本很高：

- 必须在查询期间进行许多次表达式求值
- 计算很复杂

为了改进查询的性能，可定义一个其他列，它将包含该表达式的结果。然后，当发出包括同一表达式的查询时，可直接使用生成列；或者，优化器的查询重写组件可用生成列替换该表达式。

当查询涉及连接两个或多个表中的数据时，添加生成列允许优化器选择可能更好的连接策略。将使用生成列来改进查询的性能。结果是，可能在创建和填充表之后添加生成列。

例 5-4　创建生成列的表。在 CREATE TABLE 语句上定义生成列：

```
CREATE TABLE t1(c1 INT,
                c2 DOUBLE,
                c3 DOUBLE GENERATED ALWAYS AS(c1 + c2)
                c4 GENERATED ALWAYS AS
                  (CASE WHEN c1 > c2 THEN 1 ELSE NULL END))
```

在创建此表之后，可以使用生成列来创建索引。例如：

```
CREATE INDEX i1 ON t1(c4)
```

查询可以利用生成列。例如

```
SELECT COUNT(*) FROM t1 WHERE c1 > c2
```

可以编写为：

```
SELECT COUNT(*) FROM t1 WHERE c4 IS NOT NULL
```

另一个示例

```
SELECT c1 + c2 FROM t1 WHERE(c1 + c2) * c1 > 100
```

可以编写为：

```
SELECT c3 FROM t1 WHERE c3 * c1 > 100
```

对列存在的任何<column options>进一步定义该列的属性。选项包括用于防止列包含空值的 NOT NULL，用于 LOB 数据类型的特定选项，引用类型列的 SCOPE，对表的任何约束以及列的任何默认值。

5.2.5　自动编号和标识列应用案例

标识列为 DB2 提供一种方法，可自动为添加至表的每一行生成唯一数值。当创建一个表时，如果需要将唯一标识添加至该表的每一行，那么可向该表添加一个标识列。要保证为添加至表的每一行提供唯一数字值，您应在标识列定义唯一索引，或将其声明为主键。

其他地方使用的标识列有订单号、职员编号、股票代码或者事故编号。标识列的值可以“始终”或“在默认情况下”由 DB2 数据库管理器生成。

将对定义为 GENERATED ALWAYS 的标识列给予始终由 DB2 数据库管理器生成的值。不允许应用程序提供显式的值。定义成 GENERATED BY DEFAULT 的标识列使应用程序能够显式地为标识列提供值。如果应用程序不提供值，那么 DB2 将生成一个值。因为由应用程序控制该值，所以 DB2 不能保证该值的唯一性。下面是一个 GENERATED BY DEFAULT 的示例：

```
create table db2admin.actor(
    actor_id int generated by default as identity , ----用户可以输入指定的值
    actor name varchar(20) ,
    act_yr_of_birth smallint)
```

要对新表定义标识列，在 CREATE TABLE 语句中使用 AS IDENTITY 子句。

例 5-5　创建标识列示例。在 CREATE TABLE 语句上定义标识列：

```
create table idn1(id integer, name char(20), dn integerNOT NULL
generated always as identity(start with 1, increment by 1)) ----DB2 自动生成值，用户无法输入指定值。
对于上面的表，select * from idn1 的输出为：
ID          NAME                 DN
----------- -------------------- -----------
          1 Test                 1
          2 Test                 2
          3 Test                 3
3 条记录已选择。
```

在例 5-5 中，第三个列是标识列。还可以指定该列中用来在添加行时唯一标识每一行的值：在输入的第一行的列中具有值“1”；添加到该表中的每个后续行都具有相关联的值，这些值将依次增加 1。

在上面的例 5-5 中，自动生成列的当前序列值为 4。

我们可以使用下列命令控制生成列的当前序列值：

- 执行 alter table idn2 alter column dn restart 后，当前序列值重置为 1；
- 执行 alter table idn2 alter column dn restart with 10 后，当前序列值置为 10。

5.2.6　使用 not logged initially 特性

如果我们需要经常对一个表进行批量插入、更新和删除操作，可以考虑在创建表的时候使用 not logged initially 特性。在实际生活中，这样做对于一些临时表、stage 表非常好，可以提高批量插入、更新和删除的性能。否则，如果表中数据量很大，那么批量删除、插入和更新数据时会报 SQL0964C 错误，而且也比较慢。

例 5-6　使用 not logged initially 创建表。

```
db2 => create table nolog_tab(id int, name char(20)) not logged initially
DB20000I  SQL 命令成功完成。
db2 => delete from nolog_tab ---------表中有 3 千万记录
DB21034E  该命令被当作 SQL 语句来处理，因为它不是有效的“命令行处理器”命令。在 SQL 处理期间，它返回：
SQL0964C 数据库的事务日志已满。 SQLSTATE=57011
```

对于创建表时设定了 not logged initially 的表而言，在命令行中，可以在交易中使用 alter table ... activate not logged initially 指定不记录日志。也可以考虑使用 activate not logged initially 清空表而不产生日志：

```
db2 => alter  table nolog_tab activate not logged initially with empty table
DB20000I SQL 命令成功完成。
db2 => commit
DB20000I SQL 命令成功完成。
```

当激活这个特性后，我们就可以以不记日志的方式删除表中的数据。这样不但提高了速度，也减少了日志的生成，并且减少了锁资源的使用。

我们在使用这个特性时要注意，对于一些非常重要的表，以及需要写日志的表而言，我们不建议使用该特性。

5.2.7 使用 append on 特性

在数据库中，当表中数据被删除时，空间并不会释放，而是在该行原来的位置做个“DELETED”的标志，表示该空间可以被重用。当 DB2 执行 INSERT 操作时，会扫描整个表的空闲空间并将新行置入空槽。而如果我们启用了 append on 特性，那么当插入新行时，DB2 就不必搜索空槽再插入而是直接插入到表的最后。例如：

```
CREATE TABLE appen_on_tab LIKE RECEIPTS IN SLOW_DISK_TBSP
```

可以通过将该表改变成 APPEND ON 来通知 DB2 在执行 INSERT 时不必搜索空槽：

```
ALTER TABLE appen_on_tab APPEND ON
```

这将使 INSERT 更快。这适合用于大批量追加插入一些历史表。如果启用这种特性，考虑定期 reorg 表。

5.2.8 数据、索引和大对象分开存放

在创建表的时候，考虑把表数据、索引和大对象数据分开存放到不同的表空间来提高性能。

例 5-7 创建表。

```
CREATE TABLE BOOKS( BOOKID INTEGER,
                    BOOKNAME VARCHAR(100),
                    ISBN CHAR(10) )
                    IN DATA_SPACE INDEX IN INDEX_SPACE LONG IN LONG_SPACE
```

IN、INDEX IN 和 LONG IN 子句指定将在其中存储常规表数据、索引和大对象数据的表空间。注意，这只适用于 DMS 表空间。

5.2.9　设置 pctfree

我们创建表时可以在每页上预留 pctfree%的可用空间，以应付未来的 row overflow。如果没有指定，默认预留 10%的空闲空间。通常，如果一个表中有很多 varchar 字段，当 varchar 字段更新时，如果更新的值比原来的长度长，并且在原来的行的 slot 无法存放该行数据时，这时就会在该位置留下一个指针，然后把该行插入一个新页。这样读取该行时，会造成额外的 I/O 从而影响性能。为了解决这个问题，除了定期做碎片整理(见第 11 章内容)外，还可以考虑在创建表时预留一部分空间。下面是一个使用示例：

```
CREATE TABLE RECEIPTS
    (RECEIPT DATE DATE NOT NULL, CUST NUM INT NOT NULL,
     RECEIPT_KEY TIMESTAMP NOT NULL, AMOUNT DEC(10,2),
     PRIMARY KEY(CUST NUM, RECEIPT KEY))
     PARTITIONING KEY(CUST_NUM, RECEIPT_KEY)
```

用 ALTER TABLE 语句来调整它：

```
ALTER TABLE RECEIPTS PCTFREE 10     或
ALTER TABLE RECEIPTS PCTFREE 0    -----只读表
```

注意：

假如字段 name 的数据类型为 varchar(60)，如果一开始 name 长度为 10 字节，这时假设它刚好可以放到一个数据页中。但是假设有一个 update 操作将 name 从 10 字节更新为 60 字节，如果这个数据页无法放下，那么数据库就在当前位置存放一个指针，把数据放到一个新的页中，这就叫 overflow。overflow 会增加 I/O 的读取，对性能不好。

5.2.10　表的 locksize

表的 locksize 特性与锁和并发有关，在此我们仅提醒大家在设计表的时候有这个特性，如果设计不当会严重影响应用程序并发。关于这个参数的详细解释需要结合锁和并发来讲解。我们会在“第 10 章：锁和并发”中详细讲解这个表的特性。

5.2.11　表的 volatile 特性

一些表具有下列特征：表的数据变化非常大，常常从空到非常大，又清空又变非常大。例如，我们炒股时常常需要一个交易委托单，这个交易委托单存放到一张表中。这张表在晚上做完批处理后清空为 0，第二天这个表又变得非常大。然后又清空。我们日常生活有许多诸如此类的表。对于具有这样特征的表，请启用该表的 volatile。比如在金融、银行行业需要在月末进行处理的汇总表，在不长的时间范围内数据量变化特别大，从而使

RUNSTATS 得到的统计信息不准确，原因是这些统计信息只是某个时间点的信息。您可以用下面这条语句把表修改为 volatile。

```
alter table transaction_log volatile cardinality ---设置银行交易流水表 volatile
```

这样一来，优化器将对启用 volatile 特性的表考虑使用索引扫描而不是表扫描，而无论统计信息如何。如果我们要处理的表的数据量是快速变化的，那么建议启用这个特性。

5.2.12 创建带 XML 列的表

DB2 V9 引入了一种全新的 XML 存储引擎，在这个引擎中，XML 数据是分层存储的。XML 在本质上就是分层的，所以将 XML 分层地存储在引擎中，可以保持文档的保真性，保留灵活的模式，而且能取得对子文档的较高的访问性能。这种新的分层存储引擎和关系引擎位于相同的 DB2 数据服务器中，因此现在可以将客户信息与客户的 XML 购物订单存储在一起，从而有效地搜索所有信息。

DB2 中的 XML 列

XML 以分层的格式存储在 DB2 中。XML 本身就是分层的，它从根标记(节点)开始，经历整个 XML 字符串(或文档)。在 DB2 中，XML 按照这种分层结构存储在数据页中。如果 XML 数据大于单个数据页的容量，那么 XML 树会被拆分成一些子树，每个子树存储在一个数据页中，各个页之间链接起来。

为了创建一个带 XML 数据的表，只需运行命令：

```
create table table_name(col1 data_type, ..., xml_col_name XML)
```

这样就可以创建包含您想要的关系列的表，对于 XML 信息，只需为列指定一种 XML 的数据类型。现在您可以将 XML 数据存储在那个列中。

XML 索引

创建 XML 索引与在关系数据上创建一个普通的索引类似，不同的是，您不是在一个列上创建索引，而是在前面 xml_column_name 列中定义的 XML 模式的一个组件上创建索引。其语法如下：

```
create index index name on table name(xml column name)
  generate key using xmlpattern '/po/purchaser/@pname' asSQLvarchar(50)
```

5.2.13　表维护相关命令

修改表

使用 ALTER TABLE 语句来更改列属性，例如可空性、LOB 选项、作用域、约束、压缩属性以及数据类型等等。

例如，在命令行中输入：

```
ALTER TABLE EMPLOYEE  ALTER COLUMN WORKDEPT  SET DEFAULT '123'
```

在 DB2 V9 for Linux、UNIX and Windows 上，已经对 ALTER TABLE 语句作了改进，现在它可以用来执行以下操作：

- 使用新的 DROP COLUMN 子句删除列
- 使用 ALTER COLUMN SET DATA TYPE 子句修改列属性
- 使用 SET NOT NULL 或 DROP NOT NULL 子句修改列的可空属性

在使用 SQL 修改这些表属性时，不再需要删除表并重新创建它。这原来是一个很耗费时间的过程，而且在存在对象依赖时可能会很复杂。除了上述新增加的特性外，还可以使用 DB2 V9 版本以前的修改表语句：

- 增加列。增加的新列是表中的最后一列；也就是说，如果最初有 n 列，那么添加的列将是第 n+1 列。添加新列不能使所有列的总字节数超过最大记录大小。
- 修改与列关联的默认值。在定义了新默认值后，将对任何后续 SQL 操作中指示使用此默认值的列使用新值。新值必须遵守赋值规则，且受到与 CREATE TABLE 语句下记录的限制相同的限制。

下面我们举几个使用 ALTER TABLE 语句来修改表的例子：

例 5-8　将 Managing_Bank 列添加到 ACCOUNT 表中：

```
ALTER TABLE V9R0M0.ACCOUNT ADD COLUMN Managing_Bank VARCHAR(15)
```

注意：

这个特性在 DB2 Universal Database Version 8 中就已经可用了。

例 5-9　删除 TRANSACTION 表中的 Instruction_ID 列：

```
ALTER TABLE V9R0M0.TRANSACTION DROP COLUMN Instruction_ID
```

例 5-10　将 ACCOUNT 和 TRANSACTION 表中的 Account_ID 列的数据类型从 SMALLINT 改为 INTEGER：

```
ALTER TABLE dev.ACCOUNT ALTER COLUMN Account_ID SET DATA TYPE INTEGER
ALTER TABLE dev.TRANSACTION ALTER COLUMN Account_ID SET DATA TYPE INTEGER
```

例 5-11 删除 ACCOUNT 表中的 Credit_Line 列的 NOT NULL 属性：

```
ALTER TABLE dev.ACCOUNT ALTER COLUMN Credit_Line DROP NOT NULL
```

例 5-12 增加 TRANSACTION 表中的 Description 列的大小：

```
ALTER TABLE dev.TRANSACTION ALTER COLUMN Description SET DATA TYPE VARCHAR(60)
```

注意：
这个特性在 DB2 Universal Database Version 8 中就已经可用了。

例 5-13 修改 t1 表中的 colnam1 列的默认值：

```
ALTER TABLE t1 ALTER COLUMN colnam1  SET DEFAULT '123'
```

注意：
这个特性在 DB2 V8 中就已经可用了。

重命名表

可以使用 RENAME 语句来重命名现有表。例如：

```
C:\>db2 rename table tta to rn_tab
DB20000I  SQL 命令成功完成。
```

重命名表时，源表不能在任何现有定义(视图或具体化查询表)、触发器、SQL 函数或约束中引用。它也不能具有任何生成列(标识列除外)，或者不能是父表或从属表。目录条目将更新以反映新表名。

查看表信息

可以使用表 5-1 所示的命令来获取表信息。

表 5-1　用来获取表信息的命令

命　　令	描　　述
list tables	列出用于当前用户的表
list tables for all	列出数据库中定义的所有表
list tables for schema schemaname	列出指定模式中的表

(续表)

命　　令	描　　述
list tables for schema	列出以当前用户名为模式的表
describe table tablename	显示指定的表的结构

例如，下面的命令：

```
db2 describe table department
```

产生图 5-9 所示的输出。

```
Column         Type      Type
name           schema    name           Length  Scale  Nulls
-------------  --------  -------------  ------  -----  -----
DEPTNO         SYSIBM    CHARACTER        3        0    No
DEPTNAME       SYSIBM    VARCHAR          29       0    No
MGRNO          SYSIBM    CHARACTER        6        0    Yes
ADMRDEPT       SYSIBM    CHARACTER        3        0    No
LOCATION       SYSIBM    CHARACTER        16       0    Yes
```

图 5-9　describe table degartment 命令的输出信息

删除表

可以使用 DROP TABLE 语句删除表。当删除一个表时，也会删除 SYSCAT.TABLES 系统目录中包含有关该表的信息的那一行，并会影响从属于该表的任何其他对象。例如：

- 会删除所有的列名
- 会删除基于该表的任何列创建的索引
- 将基于该表的所有视图标记为不可用
- 删除的表和从属视图的所有特权被隐式撤销
- 会删除在其中为该表父表或从属表的所有引用约束
- 从属于删除的表的所有程序包和高速缓存的动态 SQL 和 XQuery 语句被标记为无效，且该状态会保持至重新创建了从属对象为止。这包括这样的一些程序包，它们从属于将被删除的层次结构中子表上的任何超表
- 将从属于该删除表的所有触发器标记为不可用

要使用命令行来删除表，请输入：

```
DROP TABLE <table_name>
```

以下语句删除 DEPARTMENT 表：

```
DROP TABLE DEPARTMENT
```

CREATE TABLE …LIKE

如果我们想创建一个和原来表结构一样的表，可以使用 CREATE TABLE …LIKE 命令。例如，创建一个和表 employee 结构一样的表：

```
CREATE TABLE emp LIKE employee
```

获取表的 DDL 信息

可以在控制中心中，右击要导出 DDL 的表的名称，单击“生成 DDL”导出创建表的 DDL，如图 5-10 所示。

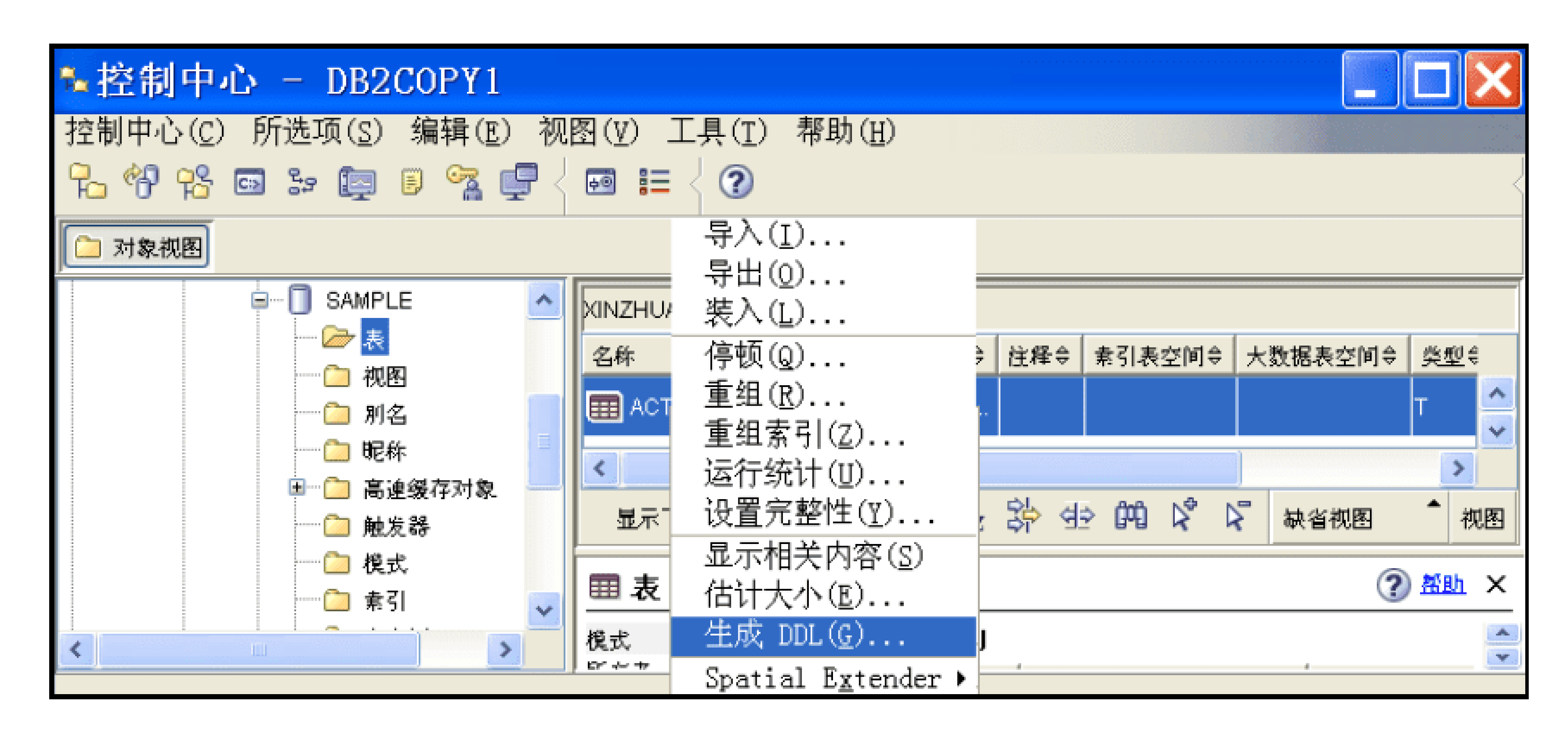


图 5-10 生成表的 DDL 信息

也可通过 db2look 命令获取创建表的 DDL，例如：

```
C:\>db2look -d sample -e -t rn_tab
-- USER 是: ORACLE
-- db2look 实用程序将只考虑指定的表
-- 正在创建表的 DDL
-- 此 CLP 文件是使用 DB2LOOK 版本创建的 9.5
-- 时间戳记: 2008-11-20 0:38:01
--数据库名称: SAMPLE
--数据库管理器版本: DB2/NT Version 9.5.0
-- 正在自动绑定程序包 ...
-- 绑定成功
CONNECT TO SAMPLE;
------------------------------------------------
```

```
--表的 DDL 语句 "ORACLE  "."RN_TAB"
------------------------------------------------
CREATE TABLE "ORACLE  "."RN_TAB" (
               "ID" INTEGERNOT NULLWITH DEFAULT1)
              IN "IBMDB2SAMPLEREL" ;
ALTER TABLE "ORACLE  "."RN TAB" PCTFREE 12;
COMMIT WORK;
CONNECT RESET;
TERMINATE;
```

注意：

db2look 命令非常强大，详细的讲解请参见“第 12 章：数据库常用工具”内容。

5.2.14　表设计高级选项

除了上面我讲的一些特性外，DB2 还有很多高级特性，例如表分区、MDC 和表压缩等。

1. 多维集群(MDC)

多维集群允许物理上同时在多个键或维上将一个表集群。在 DB2 V8 之前，DB2 只支持使用集群索引的单维数据集群。在一个表上定义一个集群索引后，当在将记录插入表中或者更新表中的记录时，DB2 试图根据集群索引的键顺序维护数据在页上的物理顺序。对于那些具有包含集群索引的键的谓词的查询，这样可以大大提高性能，因为有了良好的集群之后，就只需要访问物理表的一部分。当页面按顺序存储在磁盘上时，预取的性能会更高。

有了 MDC，相同的优点被扩展到多个维或集群键上。在查询性能方面，涉及表中一个或多个指定维的范围查询将从底层的集群获得好处。这些查询只需要访问那些包含具有指定维值的记录的页，符合条件的页将组合在一起。随着时间的推移，当表中的可用空间被填满时，具有集群索引的表可能变为非集群的。然而，一个 MDC 表可以自动、连续地在指定维上维护它的集群，而不必通过重组表来恢复数据的物理顺序。

当创建一个 MDC 表时，会指定用于顺着它们来集群表数据的维键。每个指定的维可以用一个或多个列来定义，这一点与索引键相同。对于每个指定的维，会自动创建一个维块索引，该块索引将用于快速、有效地沿着每个指定的维访问数据。此外，还会自动创建一个包含所有维键的块索引。块索引将用于维护插入和更新活动期间的数据集群，以及用于对数据进行快速有效的访问。

表的维值的每一种唯一的组合都形成了一个逻辑单元，逻辑单元在物理上由一些页块组成，每个页块是磁盘上的一组连续的页。有一些页包含的数据在某个维块索引上具有相同键值，包含这些页的一组块称作一个切片(slice)。表的每个页只存储在一个块中，表的所有块由相同数量的页组成，即所谓的分块因子(blocking factor)。分块因子与表空间的盘区大小相等，因此块边界与盘区边界成线形关系。

例 5-14 创建 MDC 表。

为了创建一个 MDC 表，需要使用 organize by 参数指定表的维，如下所示：

```
CREATE TABLE MDCTABLE(
Year INT,
Nation CHAR(25),
Colour VARCHAR(10),  ... )
ORGANIZE BY(Year, Nation, Color)
```

在这个例子中，这个表将按 Year、Nation 和 Color 这几个维来组织，逻辑上看起来如图 5-11 所示。

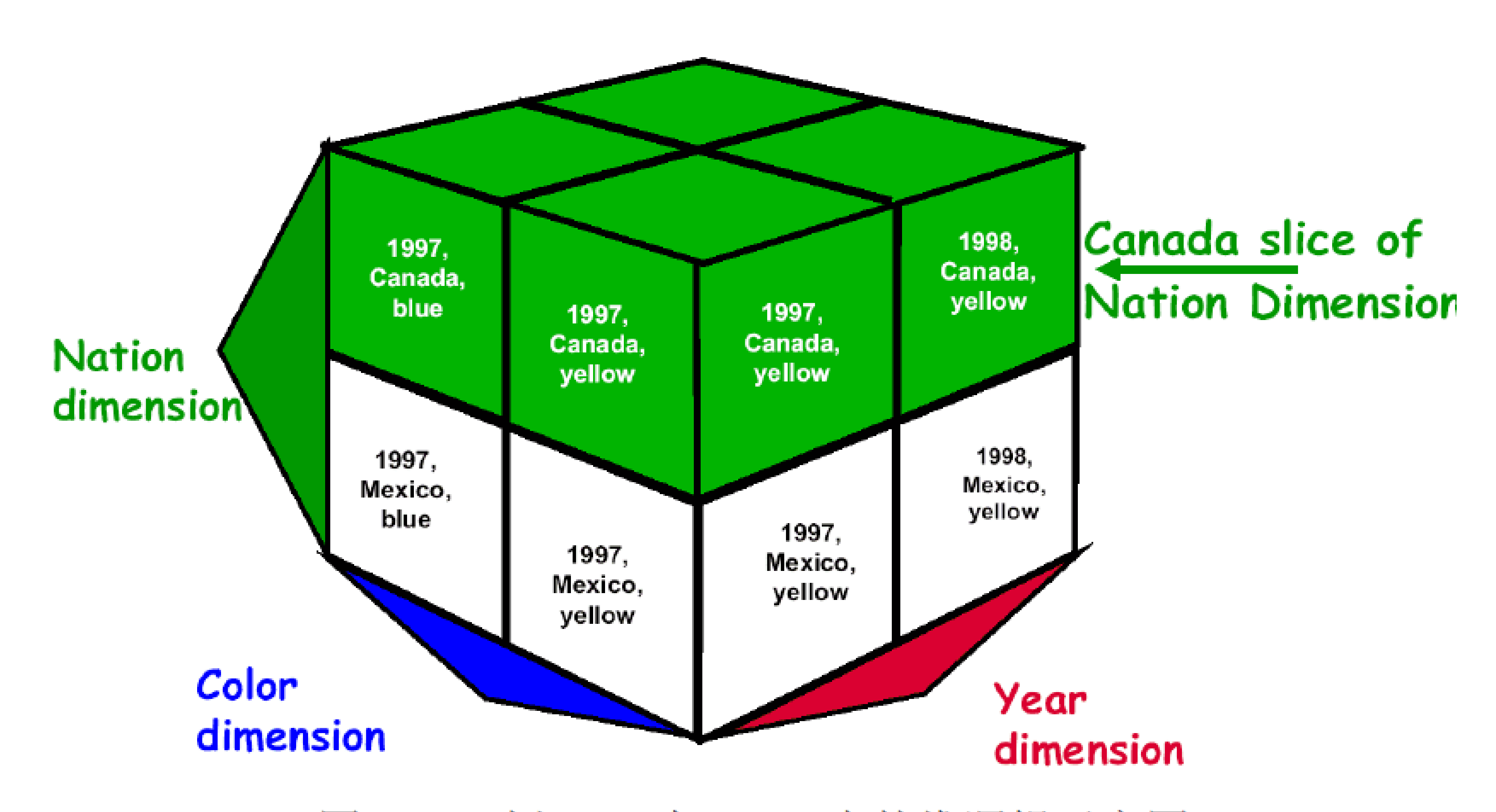


图 5-11 例 5-14 中 MDC 表的维逻辑示意图

您不能将一个表改成 MDC 表，所以在创建数据库之前，应该尽可能根据业务需求来看看您的表应该是 MDC 表还是普通的表。关于 MDC 的详细介绍，请参见《DB2 数据库性能调整和优化》中“第 4 章：数据库物理设计和逻辑设计”的内容。

2. 表(范围)分区

DB2 V8.2(及之前版本)的一些技术允许将数据拆分成更小的“块(chunk)”，以获得更大的查询并行度，消除查询中出现的分区，并帮助提高性能。如前面所讨论的，MDC 允许 DB2 安排磁盘上的数据，使具有相同维列值的行在块(一组页)中存储在一起。通过使用这

种技术，用于搜索具有特定维值的行的查询把所有其他分区排除在扫描之外，只有符合条件的行才会被访问。

类似地，数据库分区特性可以拆分一组表，使得一部分数据存放在一个数据在分区上。数据库分区可以处于不同的服务器上，这样一来，大型的扫描可以使用多个服务器的处理能力。

DB2 V9 还引入了一种新形式的分区，该特性被称为表分区(table partitioning)，它允许将单个表扩展到多个表空间上。

这种新的分区功能有很多优点，包括创建表的语法更简单。下面是一个简单的示例，创建一个分区表，用于将 24 个月的数据存储在 4 个表空间上的 24 个分区中：

```
CREATE TABLE fact
(txn_id char(7), purchase_date date, ...)
                 IN tbsp1, tbsp2, tbsp3, tbsp4
                 PARTITION BY RANGE(purchase_date)
(    STARTING FROM('2005-01-01')
ENDING('2006-12-31')
EVERY1MONTH       )
```

快速添加或删除数据范围

表分区的另一个优点是，当您分离(detach)一个分区时，可以得到一个独立的表，这个表包含了那个分区的内容。您可以将一个分区从一个表中分离出来，然后对那个新分离出来的分区做一些处理，新分离出来的分区现在实际上是一个物理表。例如，您可以归档那个表，将它移到第三存储，将它复制到另一个位置，或者做您想做的任何事情。DB2 V9 将异步地清除那个分区表上的任何索引键，而不影响正在运行的应用程序。

与添加一个新分区类似，您只需以和分区表相同的定义创建一个表，为之装入数据，然后将那个分区附加(attach)到主分区表上，如下所示：

```
ALTER TABLE FACT TABLE ATTACH PARTITION
STARTING '01-01-2007'
ENDING '01-31-2007'
FROM TABLE FACT_NEW_MONTH
```

关于表分区的详细介绍，请参见《DB2 数据库性能调整和优化》中“第 4 章：数据库物理设计和逻辑设计”的内容。

3. 表压缩

表压缩的方法是查看整个表，找到重复的字节字符串，将那些字符串存储在一个字典

中，然后用一个表示存储在字典中的实际数据的符号代替出现在表中的那些符号。其主要优点是，DB2 看到的是表中的所有数据以及完整的数据行——而不只是重复的列值。例如，如果在一个列中有一个重复的子字符串，那么可以对它进行压缩。如果多个列中存在重复的字符串(例如城市、州、人的姓名等)，那么也可以将其压缩成一个单独的符号。

要使用表压缩，首先必须对表进行设置，使之可以被压缩，然后必须生成字典，字典中包含表中出现的重复的字符串。要将表设置成可以被压缩，可以使用：

```
create table table_name ... compress yes
```

或

```
alter table tablename compress yes
```

创建压缩字典

创建压缩字典可以使表能够被压缩。DB2 需要扫描表中的数据，以发现表中出现的可以压缩的重复字符串，并将其放入字典中。为此，可以使用 reorg 命令。第一次压缩一个表(或者您想重建压缩字典)时，必须运行命令：

```
reorg table  table_name resetdictionary
```

该命令将扫描表，创建字典，然后执行实际的表重组，从而压缩数据。此后，每当插入数据到表中或者为表装载数据时，都将遵从这个压缩字典，并压缩所有新的数据。如果将来您想运行一次常规的表重组，但是不想重建这个字典，那么可以运行命令：

```
reorg table table_name keepdictionary
```

每个表都有它自己的字典，这意味着对于每个分区，分区表都有一个单独的字典。这样很有好处，因为当卷入新的分区时，DB2 能够适应数据的变化。

估计节省的空间

如果您只是想看看能节省多少空间，而不想真正对表进行压缩，那么也行。现在，DB2 INSPECT 命令有一个选项，通过该选项可以报告您决定压缩一个给定的表时可以节省的页数。语法如下：

```
db2 inspect rowcompestimate table name table_name results keep file_name
```

然后可以运行命令：

```
db2inspf file_name                      output_file_name
```

将二进制输出文件转换成一个名为 output_file_name 的文本文件。该文件包含估计通过压缩可以节省的数据页的百分比。

对一个新表进行表压缩的步骤

如果从一个新表开始，那么可能需要：

(1) 用 compress yes 创建表

(2) 将示例数据装载到表中

(3) 用 resetdictionary 重组表，以创建一个新的字典

(4) 将剩下的数据装载到表中(这次的装载将遵从上述字典，并在装载的同时进行压缩)

关于表压缩的详细介绍，请参见《DB2 数据库性能调整和优化》中“第 4 章：数据库物理设计和逻辑设计”的内容。

5.3　索引设计

5.3.1　索引优点

索引是表的一个或多个列的键值的有序列表。创建索引的原因有两个：

- 确保一个或多个列中值的唯一性。
- 提高对表进行的查询的性能。当执行查询想以更快的速度找到所需的列时，或要以索引的顺序显示查询结果时，DB2 优化器选择使用索引。如果表上不存在索引，那么必须对 SQL 查询中引用的每个表执行表扫描。表越大，表扫描所花的时间越长，因为表扫描需要顺序访问每个表行。虽然对于需要表中的大多数行的复杂查询来说，使用表扫描效率可能更高，但是对于只返回部分表行的查询而言，使用索引扫描可以更有效地访问表行。

如果在 SELECT 语句中引用了索引列，并且优化器估计索引扫描比表扫描快，那么优化器选择索引扫描。索引文件一般较小，因此读取它所需的时间比读取整个表所需的时间要少，尤其在表增大时更是如此。此外，可能不需要扫描整个索引。应用于索引的谓词减少了要从数据页读取的行数。

如果对输出的排序需求可以与索引列相匹配，那么按列顺序扫描索引将允许按正确顺序检索行而不需要排序。

每个索引条目包含一个搜索键值和一个指向包含该值的行的指针。如果在 CREATE INDEX 语句中指定了 ALLOW REVERSE SCANS 参数，那么可以按升序和降序搜索这些值。因此，在具有正确谓词的情况下，才可能对搜索分类。也可使用索引来获得已排序的

行，使数据库管理器在从表中读取这些行之后不必对它们排序。

除搜索键值和行指针外，索引还可包含(include)列，这些列是索引行中的非索引列，但是它们的数据包含在索引叶子中。这样的列有可能使优化器仅从索引获取所需要的信息，而不必访问表本身。关于 include 索引我们下面有详细的讲解。

注意：

要查询的表上存在索引并不保证结果集已排序。只有 ORDER BY 子句能确保结果集的排序。

尽管索引可显著缩短访问时间，但是它们也可给性能带来负面影响。在创建索引之前，考虑多个索引给磁盘空间和处理时间带来的影响：

- 每个索引都需要存储器或磁盘空间。准确的容量取决于表的大小以及关系索引中的列的大小和数目。
- 对一个表执行的每个 INSERT 或 DELETE 操作都需要对该表上的每个索引进行额外的更新。对于更改索引键值的每个 UPDATE 操作，也是如此。
- LOAD 实用程序重建任何现有的关系索引或追加至现有的关系索引。可在 LOAD 命令上指定 index freespace MODIFIED BY 参数，以覆盖创建索引时使用的索引 PCTFREE。每个关系索引都有可能对 SQL 查询添加备用访问路径以供优化器考虑，这会增加编译时间。

因此需要谨慎选择索引来满足应用程序的需要。

注意：

关系索引是相对于 XML 索引而言的，所以关系索引就是常规索引，也就是通常意义上的索引。在 DB2 V9 之前关系索引就是索引，DB2 V9 中有 XML 索引，为了加以区分，所以常规索引叫关系索引。

5.3.2 索引类型

有 5 种类型的索引：唯一索引、非唯一索引、集群索引、非集群索引，以及系统为多维集群(MDC)表生成的块索引。

唯一索引和非唯一索引

唯一索引是这样一种索引，它通过确保表中没有两个数据行具有完全相同的键值来帮助维护数据完整性。

尝试为已经包含数据的表创建唯一索引时，将检查组成该索引的列中的值是否唯一；如果该表包含具有重复键值的行，那么索引创建过程将失败。为表定义了唯一索引之后，每当在索引中添加或更改键时就会强制唯一性 (这包括插入、更新、装入、导入和设置完整性以命名一部分)。除了强制数据值的唯一性以外，唯一索引还可用来提高查询处理期间检索数据的性能。

另一方面，非唯一索引不用于对与它们关联的表强制执行约束。相反，非唯一索引维护频繁使用的数据值的排序顺序，这仅仅用于提高查询性能。

集群索引和非集群索引

索引体系结构分为集群或非集群。集群索引是这样的索引：数据页中的行的顺序对应于索引中的行的顺序。这就是为何给定表中只能存在一个集群索引，而表中可以存在多个非集群索引。在某些关系数据库管理系统中，集群索引的叶子节点对应于实际数据，而不是对应于指向位于其他地方的数据的指针。图 5-12 是集群索引和非集群索引的示意图。

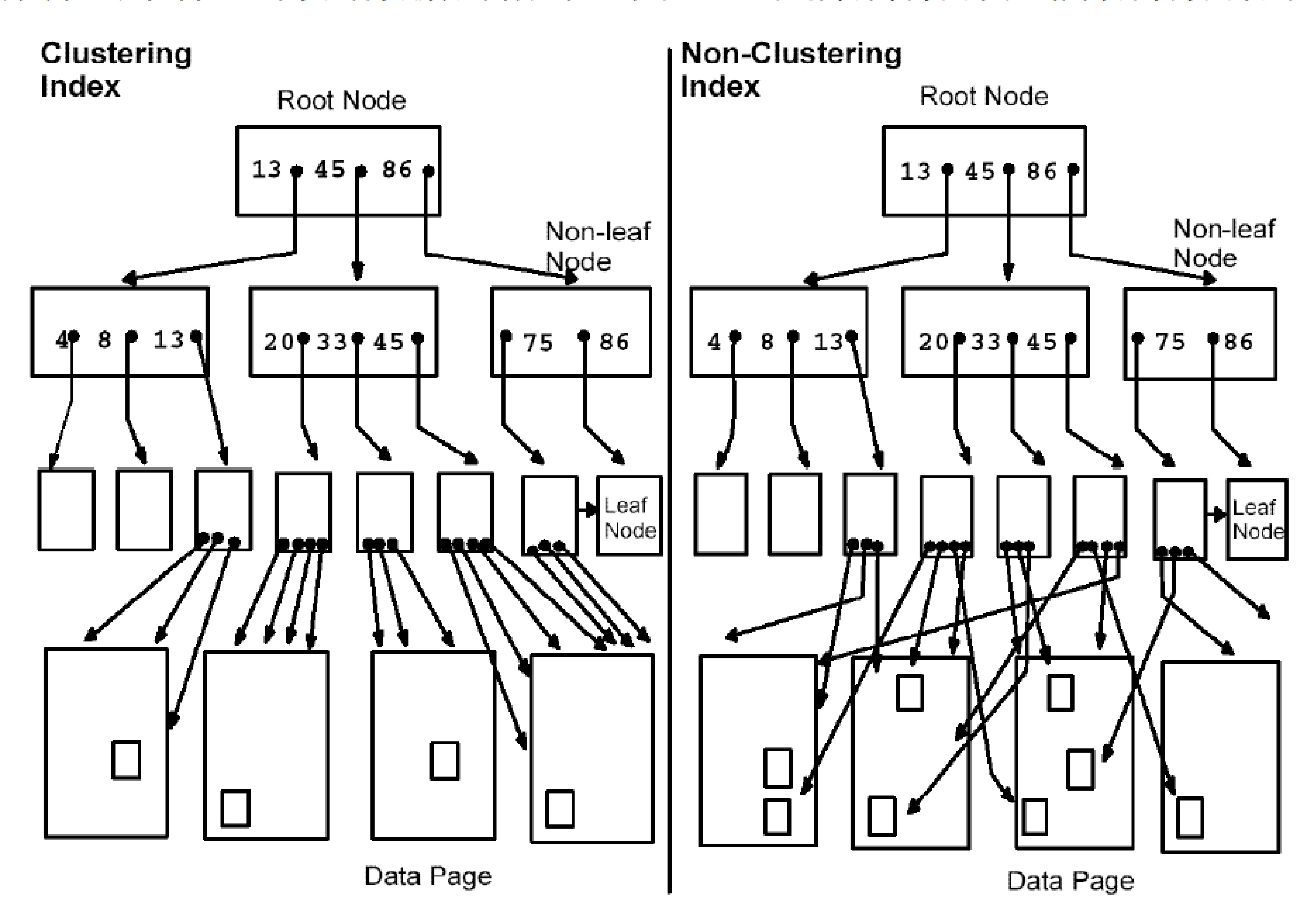


图 5-12　集群索引和非集群索引

集群索引和非集群索引都只包含索引结构中的键和记录标识。记录标识始终指向数据页中的行。集群索引和非集群索引的区别在于：数据库管理器尝试按照相应的键在索引页中的出现顺序来将数据保存在数据页中。因此，数据库管理器将尝试把具有相似键的行插入同一页中。如果对表进行了重组，那么会按照索引键的顺序将行插入数据页中。

通常，表中只有一个索引可以具有较高的集群度(因为实际的数据只可能有一种物理存放顺序)。集群索引改善了以键的顺序扫描整张表的性能。这是因为首先扫描访存第一页的第一行，然后访存同一页上的每一行，在访存了该页的所有行之后，才移至下一页。这意味着任何给定时间内都只需要表的一页位于缓冲池中。相反，如果表未集群，那么访存的每行有可能是在不同页中的。除非缓冲池中有空间保存整个表，否则这会导致每页被访存多次，从而极大地减慢扫描速度。

主键和唯一键约束与唯一索引之间的差别

了解主键和唯一键约束与唯一索引之间没有很大差别这一点很重要。数据库管理器使用唯一索引和 NOT NULL 约束的组合来实现主键约束和唯一键约束。因此，唯一索引本身不强制执行主键约束，因为它们允许空值(虽然空值表示未知值，但在建立索引时，会将一个空值视为与其他空值相同)。

因此，如果唯一索引由单个列组成，那么只允许一个空值，多个空值将违反唯一约束。同样，如果唯一索引由多个列组成，那么值和空值的特定组合只能使用一次。

双向索引

默认情况下，双向索引允许按正反两个方向进行扫描。CREATE INDEX 语句的 ALLOW REVERSE SCANS 子句同时启用正向和反向索引扫描，也就是说，按创建索引时的顺序和相反(或反向)顺序。此选项使您：

- 便于使用 MIN 和 MAX 函数
- 访存先前的键
- 不需要数据库管理器创建临时表来进行反向扫描
- 消除冗余反向顺序索引

如果指定了 DISALLOW REVERSE SCANS，那么不能反向扫描索引。

MDC 块索引

在我们创建 MDC 时，数据库会自动生成块(block)索引，它和常规行索引的区别如图 5-13 所示。

从上面的图 5-13 中我们可以看到，MDC 索引相关的索引块是排列到一起的，所以可以显著地提高性能。这部分内容本书中不做详细讲解，我们会在《深入解析 DB2》一书中深入介绍。

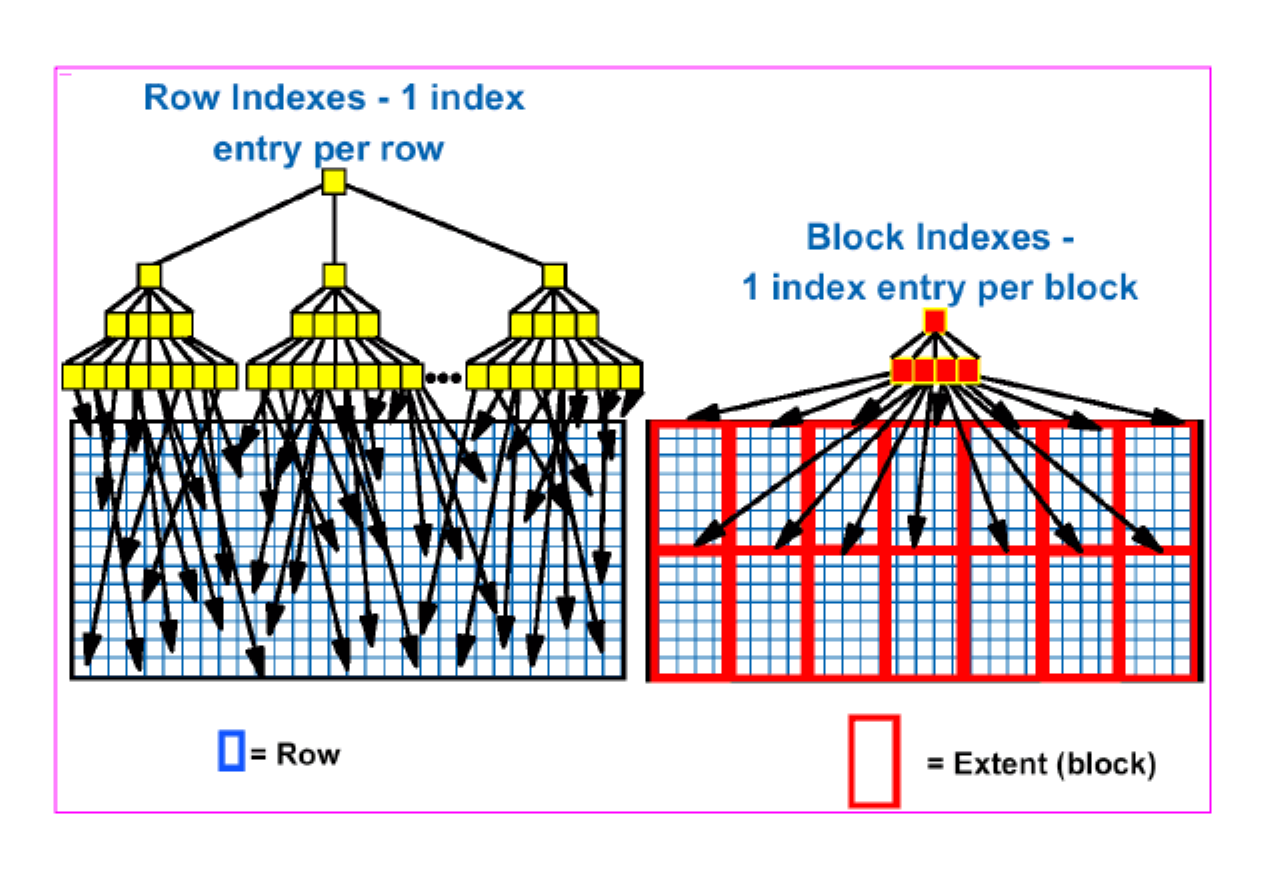


图 5-13　MDC 块索引与常规行索引的区别

5.3.3　索引结构

标准表的表和索引管理

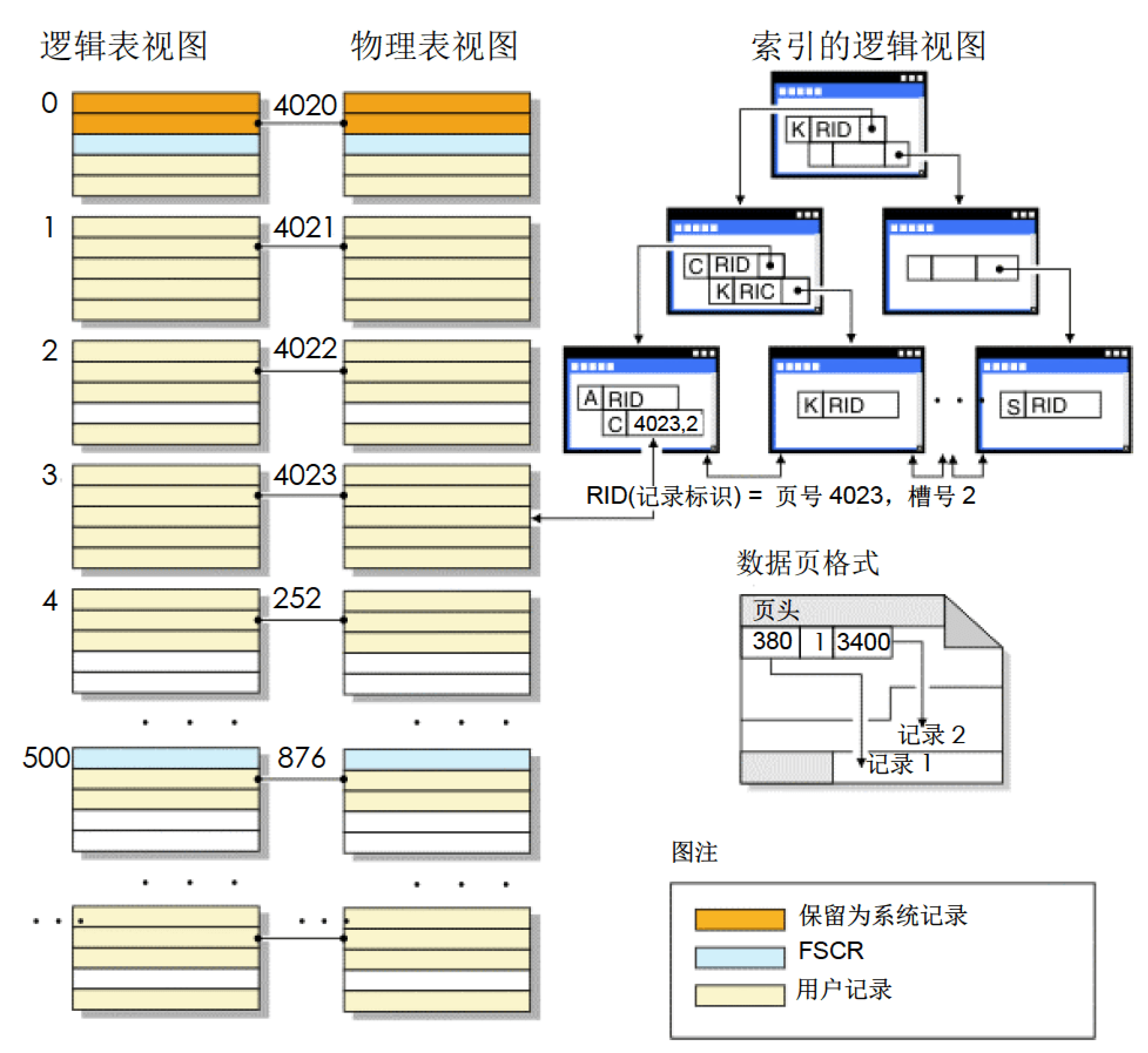


图 5-14　标准表的逻辑表、记录和索引结构

在图 5-14 中，我们可以看到在标准表中，数据在逻辑上是按数据页的列表来组织的。这些数据页根据表空间的扩展数据块大小在逻辑上分组在一起。例如，如果扩展数据块(extent)大小是 4，第 0 至 3 页是第一个扩展数据块的一部分；那么第 4 至 7 页就是第二个扩展数据块的一部分，依此类推。

根据数据页大小以及记录大小的不同，每个数据页中所包含的记录数可能会有所变化。大多数页仅包含用户记录。但是，少数页包括特殊的内部记录，DB2 使用这些记录来管理表。例如，在标准表中，每个第 500 个数据页上都有一个“空闲空间控制记录”(FSCR)。这些记录映射下面每 500 个数据页(直到下一 FSCR 为止)上的可供新记录使用的可用空间。当将记录插入表时，将使用这部分可用的可用空间。

在逻辑上，索引页组织成 B+树，这可以有效地在表中定位带有给定键值的记录。索引页上的项数不是固定的，但依赖于键的大小。对于 DMS 表空间中的表来说，索引页中的记录标识(RID)使用相对表空间页号，而不是对象相对页号。这使索引扫描能够直接访问数据页，而不需要“扩展数据块映像页”(EMP)来进行映射。

每个数据页都具有相同的格式。每个数据页开头都有一个页头。在页头后面，有一个槽目录。槽目录中的每一条目都与该页中的一个记录相对应。该条目本身是数据页中记录开始位置的字节位移。值为－1 的条目与已删除的记录相对应。

记录标识和页

记录标识(RID)由页号及随后的槽号组成。DB2 V8 后的 Type 2 索引记录还包含称为 ridFlag 的附加字段。该字段存储有关索引中密钥状态的信息，例如，此密钥是否标记为已删除。在使用索引标识了 RID 之后，便可以使用该 RID 来到达正确的数据页以及该页上正确的槽号。一旦对记录指定了 RID，在进行表重组之前，该 RID 便不会更改。数据页和 RID 的格式如图 5-15 所示。

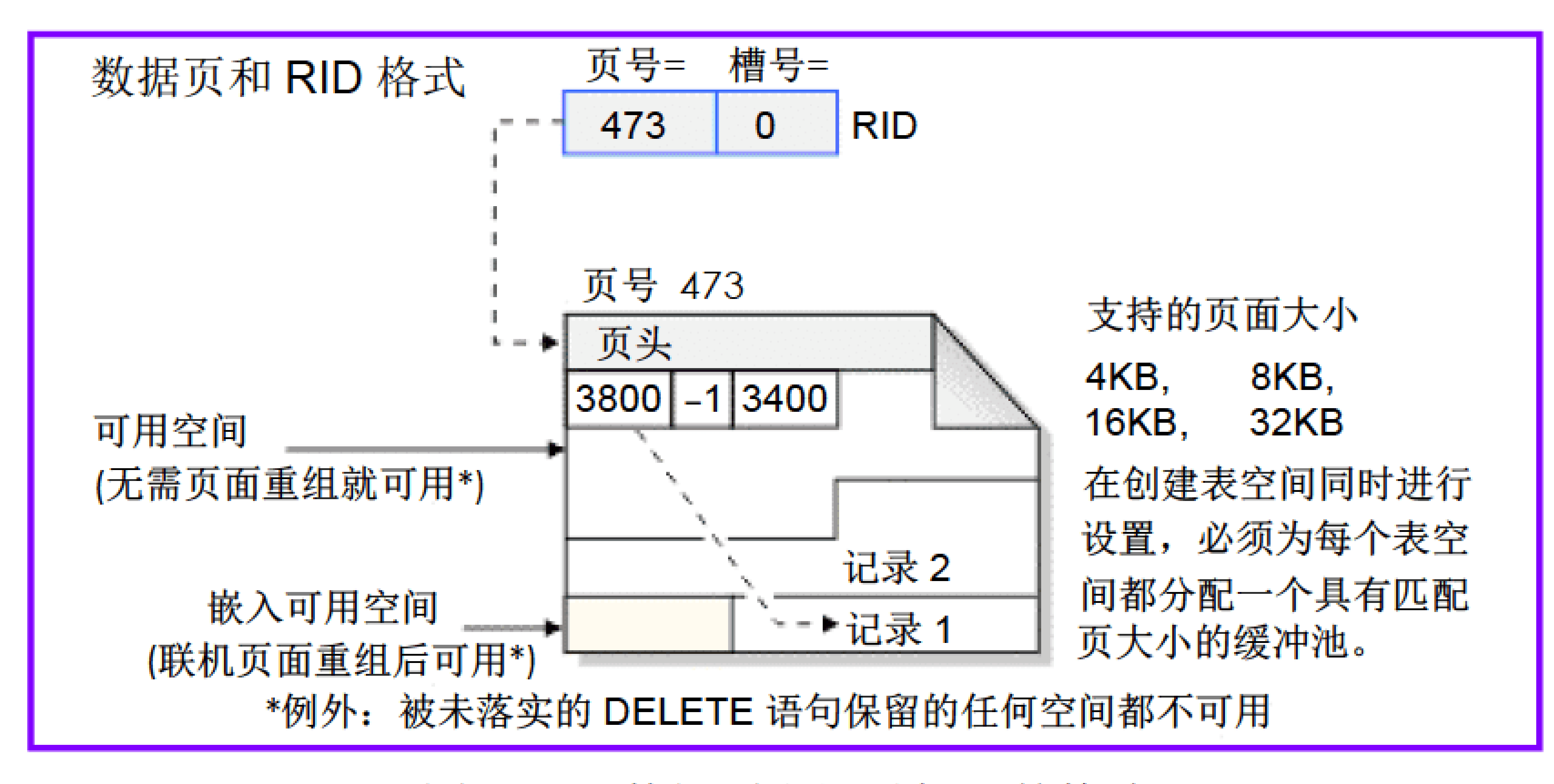


图 5-15　数据页和记录标识的格式

重组表时，实际删除记录后在页上留下的嵌入可用空间被转换成可使用的可用空间。根据记录在数据页上的移动重新定义 RID，以利用可使用的可用空间。

DB2 支持不同的页大小。对于有可能连续访问行的工作负载，请使用较大的页大小。例如，“决策支持”应用程序或大量使用临时表的场合使用的便是顺序访问。对于更有可能进行随机访问的工作负载，使用较小的页大小。例如，OLTP 环境中使用的便是随机访问。

B+树结构

DB2 数据库管理器使用 B+树结构(注：Oracle 数据库有位图索引，DB2 没有位图索引)进行索引存储。一个 B+树有一层或多层，如图 5-16 所示。其中，RID 表示行标识。

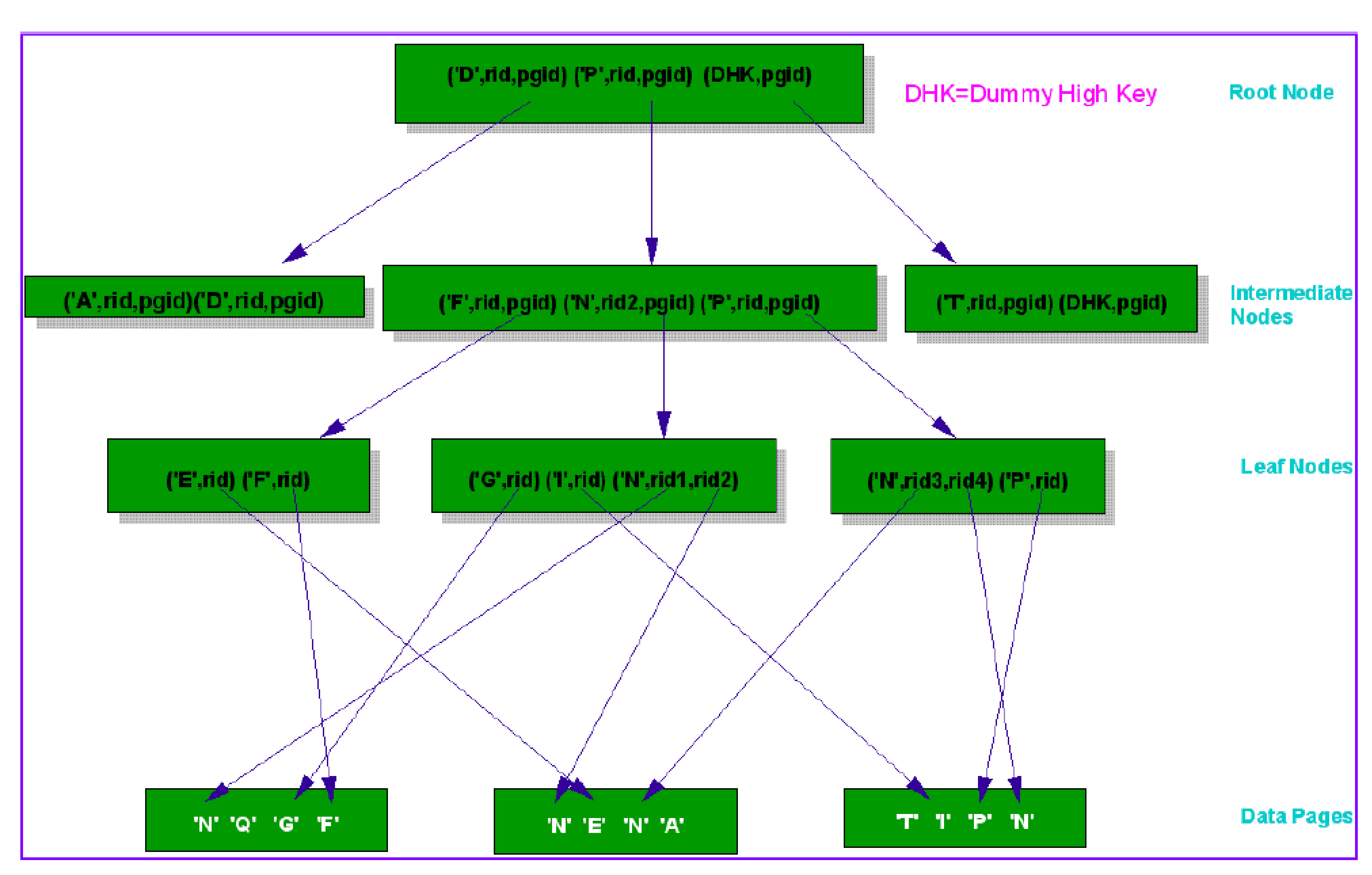


图 5-16　B+树结构

顶层称为根节点。底层由叶节点组成。底层存储了索引键值，并有一个指针(RID)指向包含键值的表中的行。根节点层和叶节点层之间的那些层称为中间节点。

当查找特定的索引键值时，索引管理器会从根节点开始搜索该索引树。对于下一层的每个节点根都包含一个键，每个键的值是下一层中对应节点的最大现有键值。例如，如果一个索引有 3 层(如图 5-16 所示)，那么，要查找一个索引键值，索引管理器搜索根节点，以查找大于或等于要查找的键的第一个键值；根节点键指向特定的中间节点。索引管理器遵循此过程遍历中间节点，直到找到包含所需要的索引键的叶节点。

5.3.4 理解索引访问机制

在优化器必须做的许多决定中，最重要的决定可能是——是否使用索引来实现查询。在优化器做此项决定之前，它必须首先确定是否存在索引。请记住，您可以查询任何表中的任何列，却不能期望单单通过索引就能做到这一点。所以，优化器必须能够访问未建立索引的数据；它可以使用扫描来做到这一点。

在大多数情形下，DB2 优化器喜欢使用索引。这是事实，因为索引可以大大优化数据检索。然而，如果不存在索引，就无法使用它了。并且在某些情况下仅仅使用数据的全扫描就可以极好地实现某些类型的 SQL 语句。例如，考虑下面这条 SQL 语句：

```
SELECT * FROM EMP;
```

在这条语句中，为什么 DB2 非要试图使用索引呢？这里没有 WHERE 子句，所以全扫描是最佳的。即使指定了 WHERE 子句，如果优化器确定页面的全表扫描要比索引扫描更好，那么也可能不会选择索引扫描这种方法。

存在索引的首要原因是它可以改善性能，那为什么有时全表扫描会比索引扫描还要好呢？这是因为索引扫描可能比简单的表扫描要慢。例如，一个非常小的表可能只有几个页面。读取所有的页面可能比先读取索引页然后再读取数据页要快。甚至对于较大的表，在某些情况下，组织索引可能需要额外的 I/O 以实现查询。当不使用索引来实现查询时，产生的存取路径会采用表扫描方法。

表扫描通常会读取表中每个页面。但在某些情况下，DB2 会非常聪明，它会限定要扫描的页面。此外，DB2 可以调用顺序预取以在请求某些页面之前就读取这些页面。当 SQL 请求需要按照数据存储在磁盘上的顺序来顺序地访问多行数据时，顺序预取特别有用。当优化器确定查询将按照顺序读取数据页面时，它会通知应该启用顺序预取。表扫描常常得益于顺序预取所作的提前预取的工作，因为当某个查询请求数据时，这些数据已经放在内存中了。

快速的索引式访问

一般来讲，访问 DB2 数据的最快方式是使用索引。索引是为了能够快速找到某个特定数据块而构造的。图 5-16 显示了 B+树索引的结构。可以看到，通过简单地从树根遍历到叶子页，可以快速地找到相应的数据页，在那里有您请求的数据。但是，DB2 所采用的索引方式因语句不同而各不相同。DB2 使用各种不同的内部算法来遍历索引结构。

在 DB2 使用索引来实现数据访问请求之前，必须满足以下条件：

- 至少有一个 SQL 谓词必须是可索引的。某些谓词因其特性而不能被索引，所以优化器不能使用索引来满足它们。

● 其中一列(在任何可索引谓词中)必须作为可用索引中的列而存在。

所以，您明白，对于 DB2，考虑使用索引的要求是相当简单的。但关于 DB2 中的索引扫描，仍有许多要了解的内容。事实上，索引扫描有各种类型。

● 直接索引查找(也是最简单的)。

对于直接索引查找，DB2 使用索引的根页面，从顶部开始，向下遍历，经过中间叶子页直到抵达相应的叶子页。在那里，它将读取实际数据页面的指针。根据索引条目，DB2 将读取正确的数据页面以返回期望的结果。对于 DB2 而言，为了执行直接索引查找，在查询的 WHERE 子句中必须为每个列提供值。例如，考虑一个 EMPLOYEE 表，该表有一个关于 DEPTNO、TYPE 和 EMPCODE 列的索引。现在考虑下面这个查询：

```
SELECT   FIRSTNAME, LASTNAMEF ROM     EMPLOYEE
WHERE    DEPTNO = 5 AND      TYPE = 'X'   AND      EMPCODE = 10;
```

如果只指定这些列中的一列或两列，则不可能采用直接索引查找，因为 DB2 没有针对每列的值，不可能匹配整个索引关键字。相反，可以选用索引扫描。有两类索引扫描：

● 匹配索引扫描和非匹配索引扫描

有时称匹配索引扫描为绝对定位；称非匹配索引为相对定位。还记得前面所讨论的表扫描吗？索引扫描与此类似。在索引扫描中，按顺序读取索引的叶子页。

匹配索引扫描从索引的根页开始，遍历至叶子页，这种扫描方式与直接索引查找方式完全一样。然而，因为无法用完整的索引关键字，所以 DB2 必须使用它所拥有的值来扫描叶子页，直到检索出所有匹配的值。现在考虑重写前面那个查询，这次没有用 EMPCODE 谓词：

```
SELECT   FIRSTNAME, LASTNAME  FROM     EMPLOYEE
WHERE    DEPTNO = 5 AND      TYPE = 'X';
```

通过从根部开始遍历索引，匹配索引扫描用相应的 DEPTNO 和 TYPE 值来查找第一个叶子页。但可能有多条索引条目具有这两个值的组合，而这些索引条目的 EMPCODE 值却不同。所以，会按顺序扫描至右边的叶子页，直到不再遇到有效的 DEPTNO、TYPE 和各种 EMPCODE 的组合。

要请求执行匹配索引，必须指定索引关键字中的高次序列，就是前面这个示例中的 DEPTNO。这向 DB2 提供了遍历索引结构的起始点，从根页开始遍历，直到相应的叶子页。但如果没有指定这个高次序列，那么会发生什么呢？假定对上面这个样本查询做点改动，不指定 DEPTNO 谓词：

```
SELECT   FIRSTNAME,  LASTNAME  FROM     EMPLOYEE
WHERE    TYPE = 'X'  AND      EMPCODE = 10;
```

在这个示例中，会用到非匹配索引扫描。在这种情形下，DB2 不能使用索引树结构，因为关键字中第一列不可用。非匹配索引扫描从索引中的第一个叶子页开始遍历，应用可用的谓词，顺序扫描后续的叶子页。不使用根页和任何中间叶子页。

- 完全索引访问(index access only)

还有一种特殊类型的索引扫描，就是“完全索引访问(index access only)”。如果所需要的全部数据都位于索引中，那么 DB2 完全可以避免读取数据页。例如：

```
SELECT   DEPTNO, TYPE
FROM     EMPLOYEE
WHERE    EMPCODE = 10;
```

请记住，我们这个数据库中包含 DEPTNO、TYPE 和 EMPCODE 列的索引。在前面的查询中，只请求查询这几列。所以，DB2 完全不需要访问表，因为在索引中可以找到所有数据。

- 多索引访问

DB2 可使用的另一类索引式访问是多索引访问。针对一个存取路径，多索引访问将使用多个索引。例如，查询 EMPLOYEE 表，其中只有两个索引：关于 EMPNO 列的 IX1 和关于 DEPTNO 列的 IX2。然后，要求这条查询显示在某个特定部门工作的员工：

```
SELECT   LASTNAME, FIRSTNME, MIDINIT
FROM     EMPLOYEE
WHERE    EMPNO IN('000100', '000110', '000120')
AND      DEPTNO = 5;
```

DB2 将会使用用于 EMPNO 谓词的 IX1 还是使用用于 DEPTNO 谓词的 IX2？为什么不一起使用这两者呢？这就是多索引访问的实质所在。根据谓词是用 AND 连接还是用 OR 连接，可将多索引访问分为两类：IndexANDING 和 IndexORING。IndexANDING 是先使用索引 IX1 和 IX2 取到索引扫描的结果，然后对两个扫描结果取交集；而 IndexORING 是先使用索引 IX1 和 IX2 取得索引扫描结果后，然后执行合并操作。

5.3.5 创建集群索引

以下 SQL 语句在 EMPLOYEE 表的 LASTNAME 列上创建一个群集索引，称为 INDEX1：

```
CREATE INDEX INDEX1 ON EMPLOYEE(LASTNAME) CLUSTER
```

为了让语句更有效，可以通过与 ALTER TABLE 语句相关的 PCTFREE 参数来使用群集索引，以便于将新数据插入到正确的页上，从而维护该群集的次序。通常情况下，表上的 INSERT 操作越多，为维护群集所需要的 PCTFREE 值就越大。因为这个索引确定数据

在物理页上放置的次序，所以对任何特定的表只能定义一个群集索引。

另一方面，如果这些新行的索引关键字值总是新的大关键字值，那么表的群集属性将尝试把它们放到表的末尾。其他页上有空闲空间对保持群集没有什么作用。在这种情况下，将表设置为追加方式可能优于使用群集索引，改变表来拥有一个大的 PCTFREE 值。可以执行如下命令来将表设置为追加方式：ALTER TABLE APPEND ON。

以上讨论也适用于增大行大小的 UPDATE 引起的新的“溢出(overflow)”行。

5.3.6　创建双向索引

一个使用 CREATE INDEX 语句中的 ALLOW REVERSE SCANS 参数创建的单索引可以向左或者向右扫描。也就是说，这些索引支持按照在反方向创建和扫描索引时所定义的方向索引。这个 SQL 语句如下。

```
CREATE INDEX iname ON tname(cname DESC) ALLOW REVERSE SCANS
```

在这种情况下，基于给定列(cname)中的递减值(DESC)形成索引(iname)。尽管列上的索引定义用来按照递减次序扫描，通过允许反向扫描，可以按照降序(反向)扫描。实际上没有使用这两个方向上的索引，创建和考虑存取模式时由优化器控制这些索引的使用。

索引页合并与分裂

CREATE INDEX 语句的 MINPCTUSED 子句指定在索引叶页上最小已用空间的阈值。如果使用这个子句，那么可以对这个索引启用联机索引重组。一旦启用了联机索引重组，就可以参照以下考虑事项来确定是否执行联机重组：当从这个索引的一个索引叶子页(leaf)中删除一个关键字(key)后，如果该页上已用空间的百分比小于所指定的阈值，那么就检查相邻的索引叶页来确定是否可以将两个叶页上的关键字合并到单个索引页中。例如，下列 SQL 语句创建启用联机索引重组的索引。

```
CREATE INDEX LASTN ON EMPLOYEE(LASTNAME) MINPCTUSED 20
```

当从这个索引删除一个关键字时，如果这个索引页上的其余关键字占用索引页上 20%或更小的空间，那么就可以尝试将这个索引页的关键字与相邻索引页的关键字合并，来删除这个索引页。如果组合的关键字可以全部位于一页上，那么就执行这个合并，并删除其中一个索引页。

图 5-17 是设置了 MINPCTUSED 后索引在线重组的示意图。

CREATE INDEX 语句的 PCTFREE 子句指定，创建索引时每个索引页中要留作空闲空间的百分比。在索引页上保留更多的空闲空间将导致更少的页分割，这将减少为重新获得顺序索引页面而重组表的需要，从而增加预存取，而预存取是一个可以提高性能的重要部

件。此外，如果总是存在大关键字值，那么就要考虑降低 CREATE INDEX 语句的 PCTFREE 子句的值。

对于只读表上的索引，使 PCTFREE 为 0；对于其他索引，使 PCTFREE 为 10，以提供可用的空间，从而加快插入操作的速度。此外，对于有群集索引的表而言，这个值应该更大一些，以确保群集索引不会被分成太多的碎片。如果存在大量的插入操作，那么使用 15 到 35 之间的值或许更合适一些。

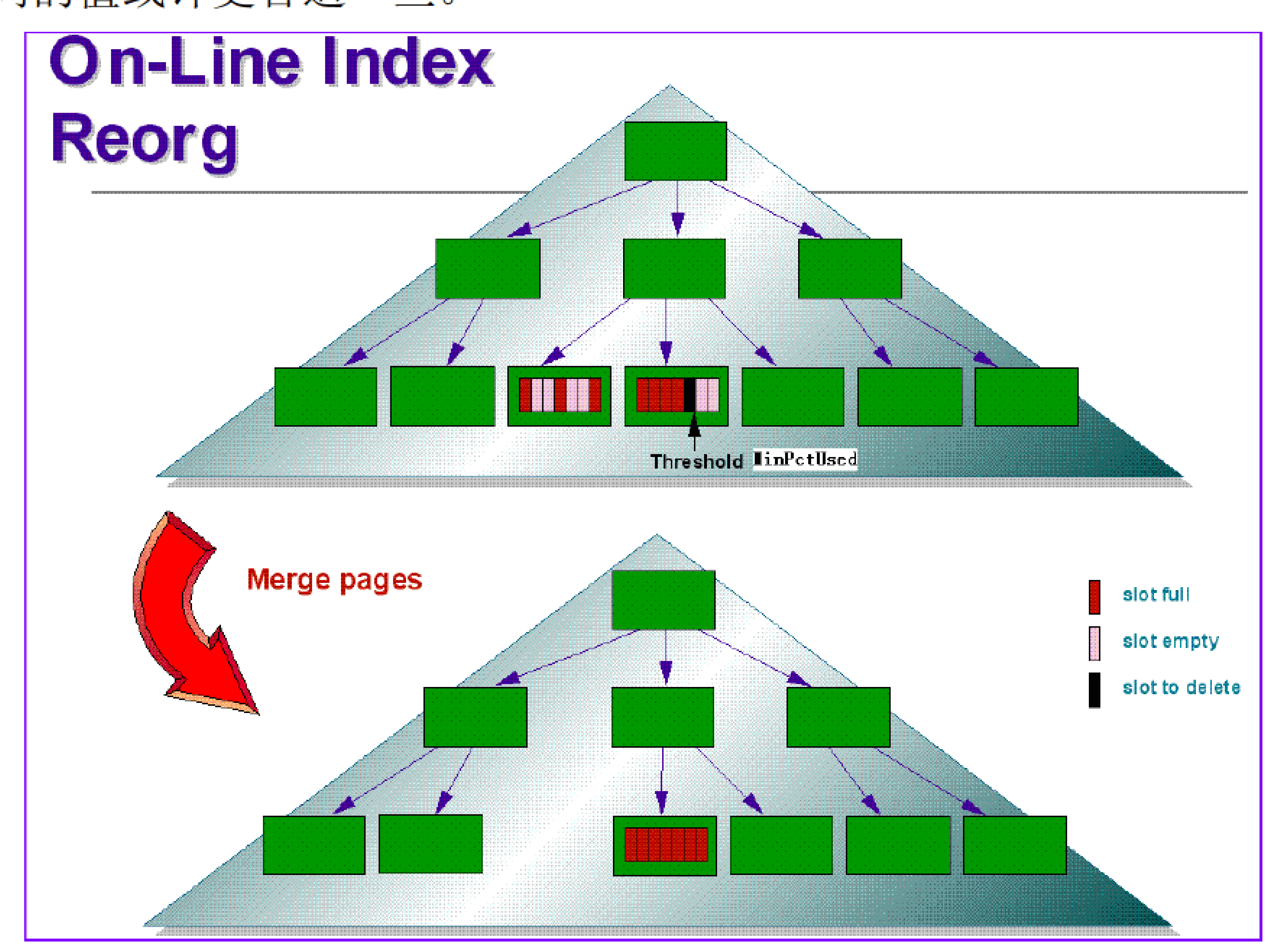


图 5-17　索引的在线重组

5.3.7　完全索引访问(index access only)

CREATE INDEX 语句的 INCLUDE 子句指定在创建索引时，可以选择包含附加的列数据，这些附加的列数据将与键存储在一起，但实际上它们不是键自身的一部分，所以不被排序。在索引中包含附加列的主要原因是为了提高某些查询的性能。DB2 将不需要访问数据页，因为索引页早已经提供了数据值。只可以为包含的列定义唯一索引。但在强制执行索引的唯一性时不考虑被包含的列。

假设我们经常需要获得按 EMPNO 排序的员工列表。查询将如下所示：

```
SELECT EMPNO,EMPNAME FROM EMP ORDER BY EMPNO
```

下面的语句会创建一个可以提高性能的可能的索引：

```
CREATE UNIQUE INDEX IEMPNO ON EMPNO(EMPNO) INCLUDE(EMPNAME)
```

结果，查询结果所需的所有数据都显示在索引中，不需要检索数据页。那么，为什么不干脆在索引中包括所有的数据呢？首先，这需要数据库中的更多物理空间，因为本质上数据是在索引中复制的。其次，只要更新了数据的值，数据的所有副本都需要更新，在发生许多次更新的数据库中，这是一项很大的开销。

5.3.8　创建索引示例

在最频繁处理的查询和事务的 WHERE 子句中所使用的那些列上创建关系索引。

以下的 WHERE 子句

```
WHERE WORKDEPT='A01' OR WORKDEPT='E21'
```

通常会从 WORKDEPT 上的索引获益，除非 WORKDEPT 列包含许多重复值。

在按查询所需要的顺序对行排序的一列或多列上创建关系索引时，不仅在 ORDER BY 子句中，而且其他功能，如 DISTINCT 和 GROUP BY 子句也都需要排序。

以下示例使用 DISTINCT 子句：

```
SELECT DISTINCT WORKDEPT          FROM EMPLOYEE
```

数据库管理器可使用 WORKDEPT 上定义为升序或降序的索引来消除重复值。此同一个索引也可用于 GROUP BY 子句中，以将值分组，如下所示：

```
 SELECT WORKDEPT, AVERAGE(SALARY)
 FROM EMPLOYEE     GROUP BY WORKDEPT
```

使用复合键创建索引，该键命名语句中引用的每个列。当用此方式指定索引时，可以从完全索引检索关系数据，这比访问表更有效。

例如，考虑下列 SQL 语句：

```
SELECT LASTNAME  FROM EMPLOYEE  WHERE WORKDEPT IN('A00','D11','D21')
```

如果为 EMPLOYEE 表的 WORKDEPT 和 LASTNAME 列定义了关系索引，那么通过扫描索引而不是扫描整个表可能会更有效地处理该语句。注意，因为该谓词基于 WORKDEPT，因此此列应是该关系索引的第一列。

使用 INCLUDE 列创建关系索引可改善表上索引的使用。使用上述示例，可将唯一关系索引定义为：

```
CREATE UNIQUE INDEX x ON employee(workdept) INCLUDE(lastname)
```

指定 lastname 为 INCLUDE 列而不是索引键的一部分，意味着 lastname 只存储在索引的叶子页上。

1. 根据查询所使用的列建立索引

建立索引是用来提高查询性能最常用的方法。对于一个特定的查询，可以为某一个表所有出现在查询中的列建立一个复合索引，包括出现在 SELECT 语句和条件子句中的列。但简单地建立一个覆盖所有列的索引并不一定能有效提高查询，因为在多列索引中列的顺序是非常重要的。这个特性是由索引的 B+树结构决定的。一般情况下，要根据谓词的选择度来排列索引中各列的位置，选择度大的谓词所使用的列放在索引的前面，把那些只存在于 SELECT 语句中的列放在索引的最后。例如下面的查询：

例 5-15 索引中的谓词位置。

```
select add_date  from temp.customer  where city = 'WASHINGTON'
and cntry_code = 'USA';
```

对于这样的查询，可以在 temp.customer 上建立(city，cntry_code，add_date)索引。由于该索引包含了 temp.customer 所有用到的列，所以此查询将不会访问 temp.customer 的数据页面，而是直接使用了索引页面。对于包含多列的复合索引，索引树中的根节点和中间节点存储了多列的值的联合。这就决定了存在两种索引扫描。回到例 5-15 中的查询，由于此查询在新建索引的第一列上存在谓词条件，所以 DB2 能够根据这个谓词条件从索引树的根节点开始遍历，经过中间节点，最后定位到某一个叶子节点，然后从此叶子节点开始往后进行在叶子节点上的索引扫描，直到找到所有满足条件的记录。这种索引扫描就是我们前面所讲的匹配索引扫描(Matching Index Scan)。但是如果将 add_date 放在索引的第一个位置，而查询并不存在 add_date 上的谓词条件，那么这个索引扫描将会从第一个索引叶子节点开始，它无法从根节点开始并经过中间节点直接定位到某一个叶子节点，这种扫描的范围扩大到了整个索引，这就是前面所提到的非匹配索引扫描(Non-Matching Index Scan)。图 5-18 显示了 DB2 根据不同索引生成的存取计划。

2. 根据条件语句中的谓词的选择度创建索引

因为建立索引需要占用数据库的存储空间，所以需要在空间和时间性能之间进行权衡。很多时候，只考虑那些在条件子句中有条件判断的列上建立的索引也会同样有效，同时节约了空间。例如例 5-15 中的查询，可以只建立(city，cntry_code)索引。我们还可以进一步地检查条件语句中的这两个谓词的选择度：

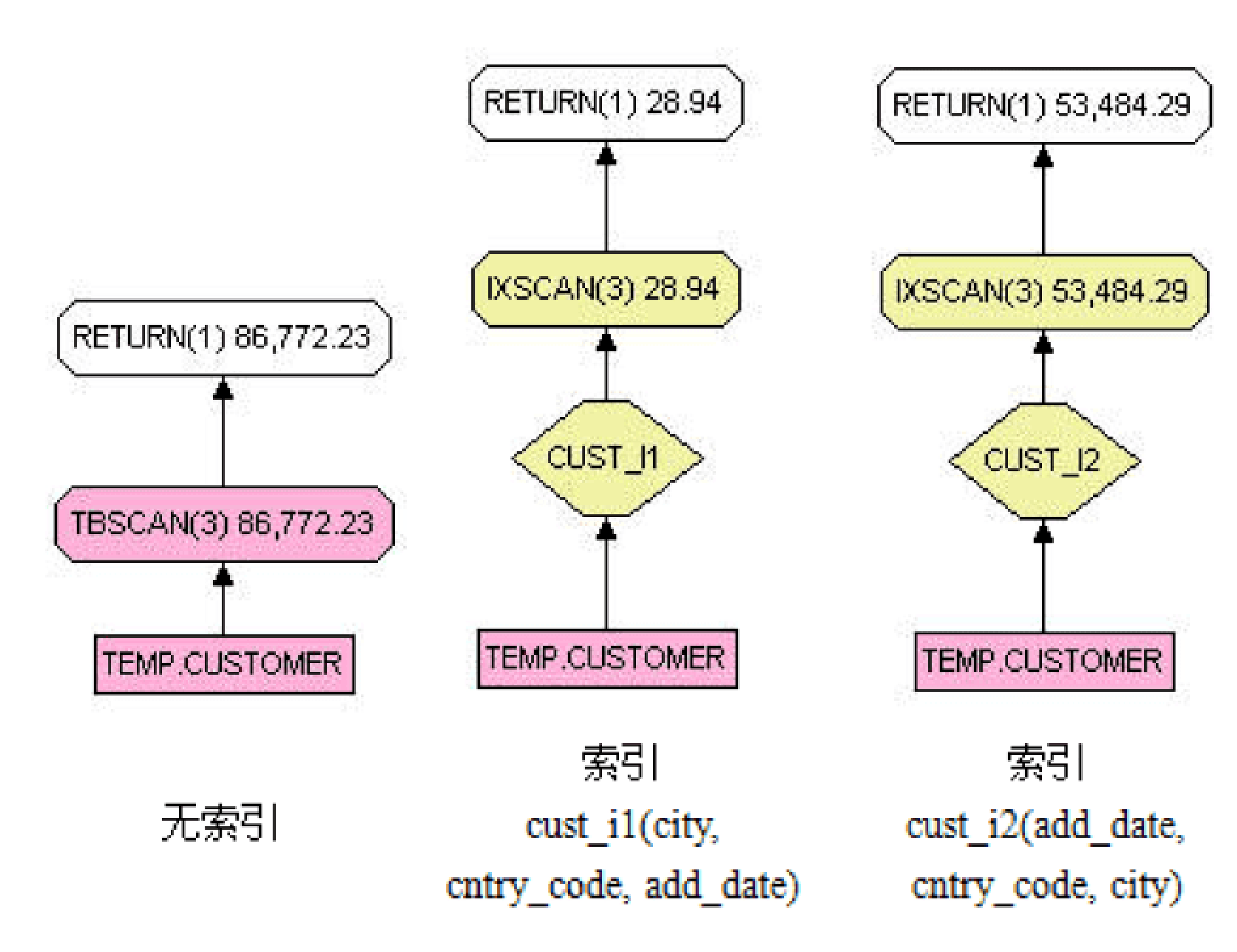


图 5-18　根据不同索引生成的存取计划

查询:

```
1. select count(*) from temp.customer where city = 'WASHINGTON'
and cntry_code = 'USA';
2. select count(*) from temp.customer where city = 'WASHINGTON';
3. select count(*) from temp.customer where cntry_code = 'USA';
```

Results:

```
1. 1404
2. 1407
3. 128700
```

选择度越大，过滤掉的记录越多，返回的结果集也就越小。从上面的结果可以看到，第二个查询的选择度几乎和整个条件语句相同。因此可以直接建立单列索引(city)，其性能与索引(city，cntry_code，add_date)相差不多。表 5-2 对这两个索引的性能和大小进行了对比。

表 5-2　两个索引的性能和大小对比

索　　引	查询计划总代价	索 引 大 小
cust_i1(city，cntry_code，add_date)	28.94 timerons	19.52MB
cust_i3(city)	63.29 timerons	5.48MB

从表 5-2 中可以看到，单列索引(city)具有更加有效的性能空间比，也就是说占有尽可

能小的空间而得到尽可能高的查询速度。

3. 避免在建有索引的列上使用函数

这是一个很简单的原则，如果在建有索引的列上使用函数(函数的单调性不确定，函数的返回值和输入值可能不会一一对应)，那么就可能存在索引中位置差异很大的多个列值可以满足带有函数的谓词条件，DB2 优化器将无法进行 Matching Index Scan，更坏的情况下可能会导致直接进行表扫描。图 5-19 对比了使用 function 前后存取计划的变化。

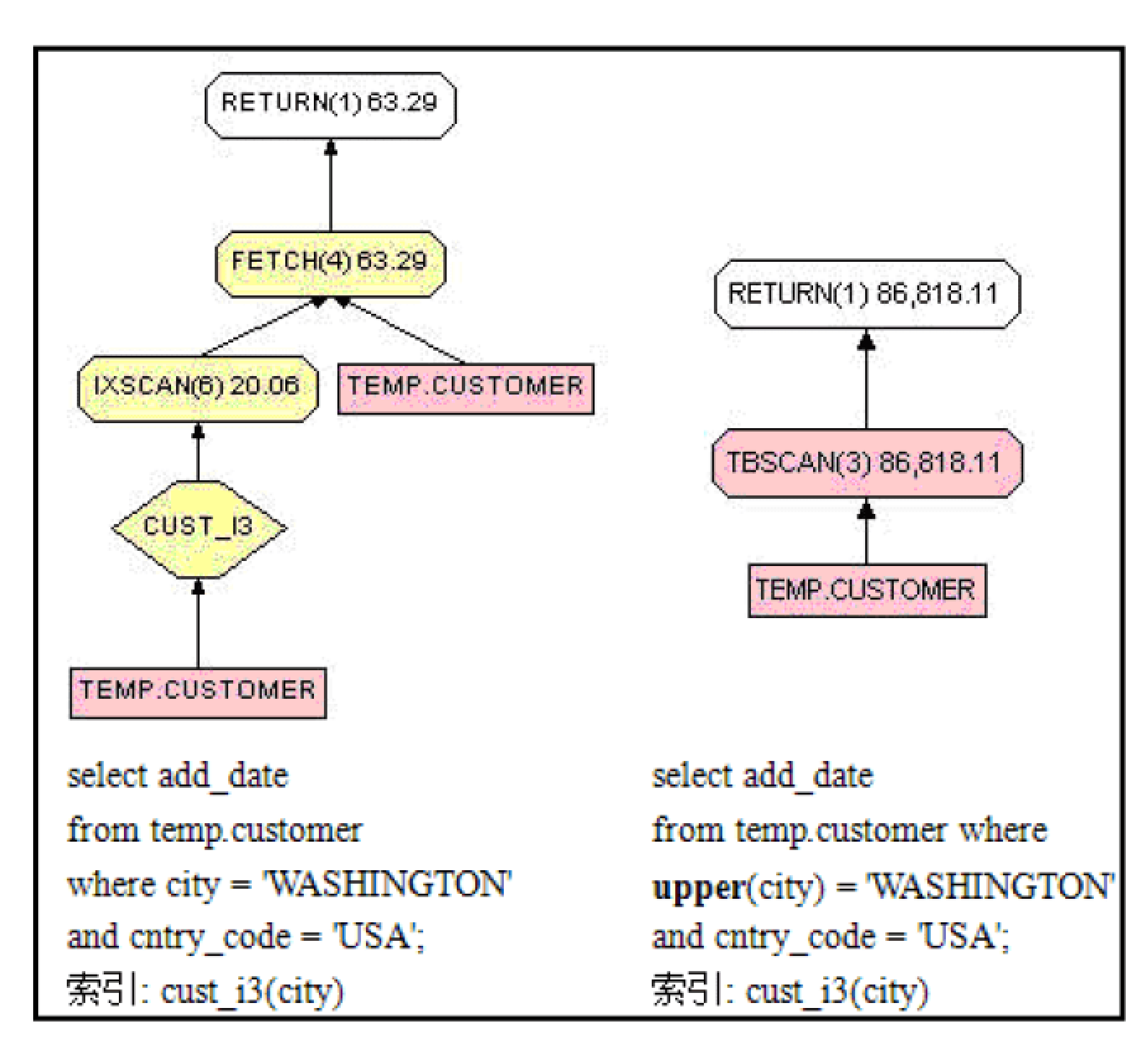


图 5-19　使用 function 前后存取计划的变化

4. 在那些需要被排序的列上创建索引

这里的排序不仅仅指 ORDER BY 子句，还包括 DISTINCT、UNION 和 GROUP UP 子句，它们都会产生排序操作。由于索引本身是有序的，在其创建过程中已经进行了排序处理，因此在应用这些语句的列上创建索引会降低排序操作的代价。这种情况一般针对没有条件语句的查询。如果存在条件语句，DB2 优化器会首先选择出满足条件的记录，然后才对中间结果集进行排序。对于没有条件语句的查询，排序操作在总的查询代价中会占有较大比重，因此能够较大限度地利用索引的排序结构进行查询优化。此时可以创建单列索引，如果需要创建复合索引，则需要把被排序的列放在复合索引的第一列。图 5-20 对比了例 5-16 中的查询在创建索引前后的存取计划。

例 5-16　查询在创建索引前后的存取计划。

```
Select distinct add_date from temp.customer
```

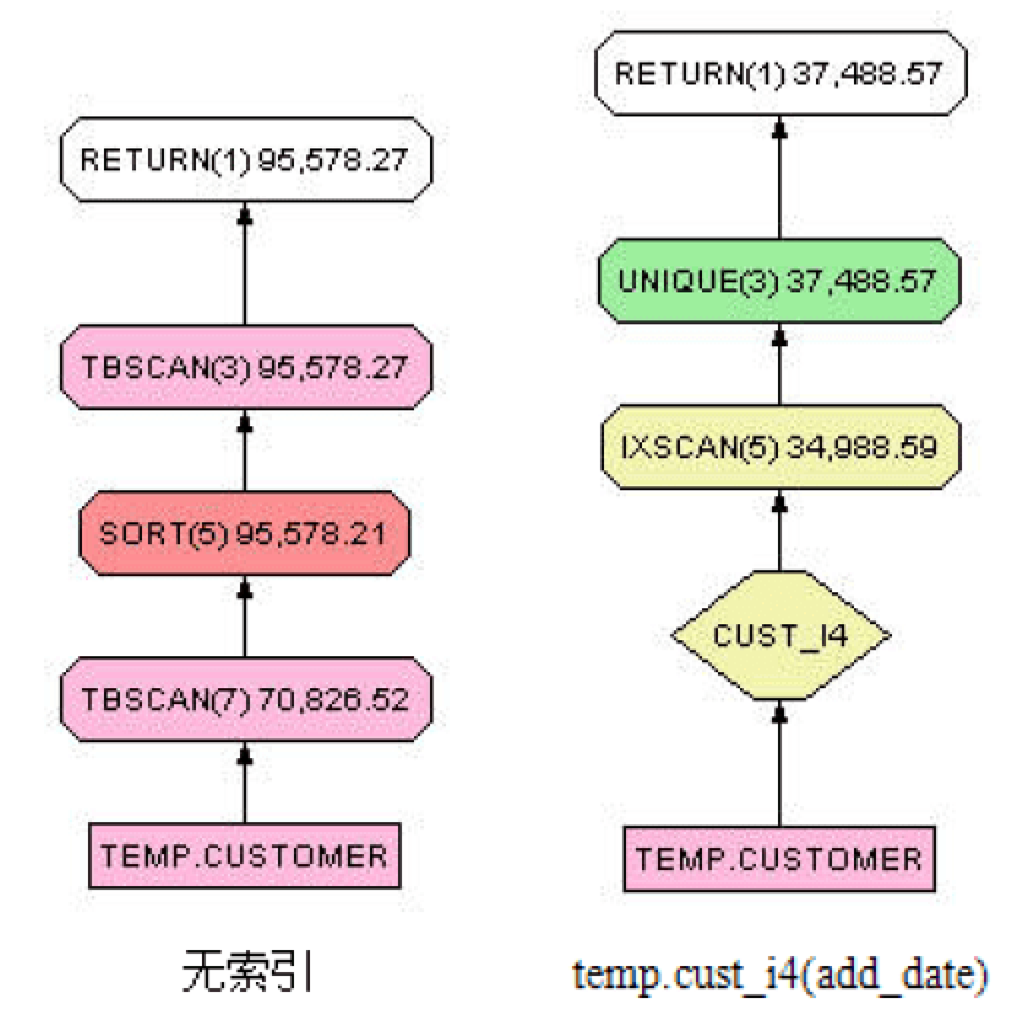


图 5-20　在创建索引前后的存取计划

从图 5-20 中我们可以看到，在没有索引的情况下，SORT(排序)操作是 95578.21 timerons；但是在有索引的情况下，不再需要对结果集进行排序，可以直接进行 UNIQUE 操作，图中显示这一操作只花费了 37488.57 timerons.

图 5-21 对比了以下查询在创建复合索引前后的存取计划，从中可以更好地理解索引对排序操作的优化。

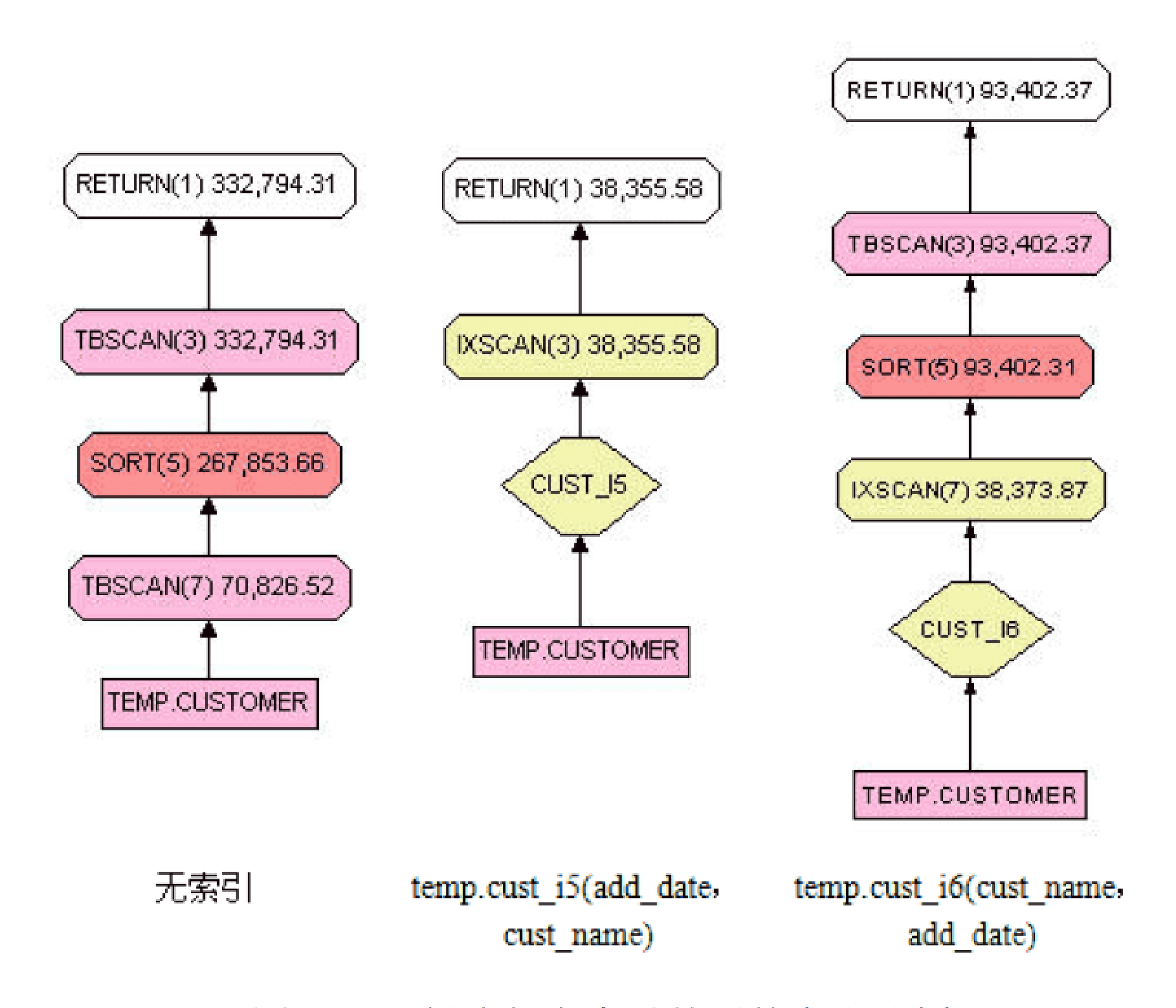


图 5-21　创建复合索引前后的存取计划

```
select cust_name from temp.customer order by add_date;
```

索引的 B+树结构决定了索引 temp.cust_i5 的所有叶子节点本身就是按照 add_date 排序的，所以对于上面的查询，只需要顺序扫描索引 temp.cust_i5 的所有叶子节点。但是对于 temp.cust_i6 索引，其所有叶子节点是按照 cust_name 排序的，因此在经过对索引的叶子节点扫描，并获得所有数据之后，还需要对 add_date 进行排序操作。

5. 合理使用 include 关键词创建索引

对于类似下面的查询：

```
select cust name from temp.customer
where cust_num between '0007000000' and '0007200000'
```

在前面我们提到，可以在 cust_num 和 cust_name 上建立复合索引来提高查询性能。但是由于 cust_num 是主键，所以可以使用 include 关键字创建唯一性索引：

```
create unique index temp.cust_i7 on temp.customer(cust_num) include(cust_name)
```

使用 include 后，cust_name 列的数据将只存在于索引树的叶子节点，而不存在于索引的关键字中。这种情况下，使用带有 include 列的唯一索引会带来优于复合索引的性能，因为唯一索引能够避免一些不必要的操作，如排序。对于上面的查询，创建索引 temp.cust_i7 后存取计划的代价为 12338.7 timerons，创建复合索引 temp.cust_i8(cust_num，cust_name) 后的代价为 12363.17 timerons。一般情况下，当查询的 WHERE 子句中存在主键的谓词，我们就可以创建带有 include 列的唯一索引，形成纯索引访问来提高查询性能。注意 include 只能用在创建唯一性索引中。

6. 指定索引的排序属性

下面是一个用来显示最近一个员工入职的时间的查询：

```
select max(add_date) from temp.employee
```

很显然这个查询会进行全表扫描。查询计划如图 5-22(a)所示。

显然我们可以在 add_date 上创建索引。根据下面命令创建索引后的查询计划如图 5-22(b)所示。

```
create index temp.employee_i1 on temp.employee(add_date)
```

这里存在一个误区，大家可能认为：既然查询里要取得的是 add_date 的最大值，而我们又在 add_date 上建立了一个索引，优化器应该知道从索引树中直接去寻找最大值。但是

实际情况并非如此，因为创建索引的时候并没有指定排序属性，默认为 ASC 升序排列，DB2 将会扫描整个索引树的叶子节点并取得所有值后，取其最大值。我们可以通过设置索引的排序属性来提高查询性能，根据下面命令创建索引后的查询计划如图 5-22(c)所示。

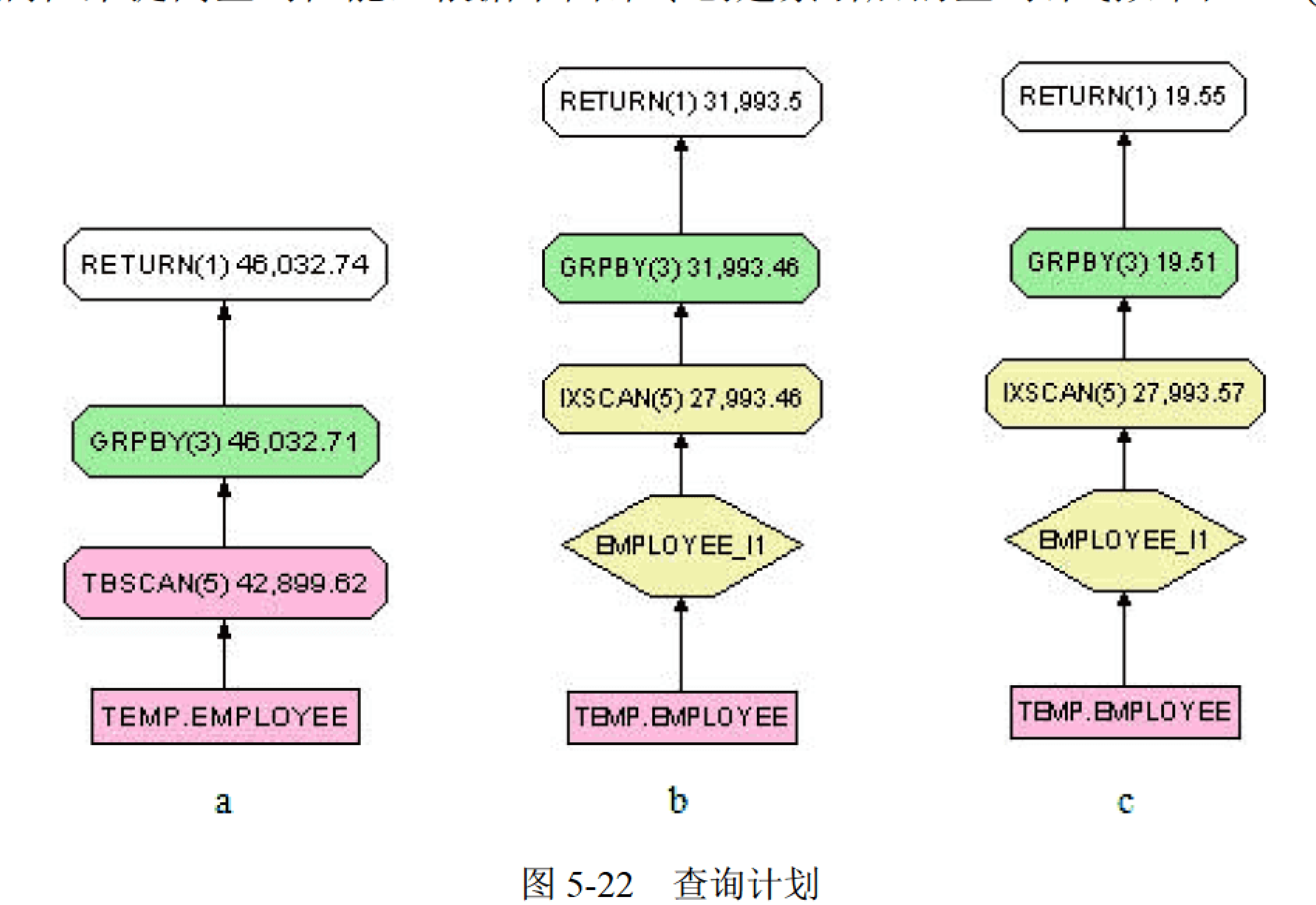


图 5-22　查询计划

```
    create index temp.employee_i1 on temp.employee(add_date desc)
```

对于降序排列的索引，DB2 不需要扫描整个索引数的叶子节点，因为第一个节点便是最大的。我们同样可以使用 ALLOW REVERSE SCANS 来指定索引为双向扫描，具有和 DESC 近似的查询性能。ALLOW REVERSE SCANS 可以被认为是 ASC 和 DESC 的组合，只是在以后数据更新的时候维护成本会相对高一些。

如果无法改变索引的排序属性，但是我们具有额外的信息，如该公司每个月都会有新员工入职，那么这个查询就可以改写成：

```
    select max(add date) from temp.employee where add date > current timestamp
-1 month
```

这样一来，通过限定一个查询范围也会有效地提高查询性能。

5.3.9　索引总结

创建的索引应该取决于数据和存取该数据的查询。

以下准则可帮助您确定如何创建可用于各种目的的索引：

- 要避免某些排序，只要有可能，就使用 CREATE UNIQUE INDEX 语句定义主键和唯一键。
- 要改善数据检索，将 INCLUDE 列添加至唯一索引。合适的列为：
  ——被频繁存取，因此可从完全索引访问(index access only)受益的列
  ——不需要用来限制索引扫描的范围的列
  ——不影响索引键的排序或唯一性的列
- 要有效存取小表，使用索引来优化对含有较多数据页的表的频繁查询，数据页数记录在 SYSCAT.TABLES 目录视图的 NPAGES 列中。您应该：
  ——根据连接表时要使用的任何一列来创建索引
  ——根据将用于定期索引特定值的任何列来创建索引
- 要有效地搜索，对键使用升序还是降序取决于将最常使用的次序。尽管当在 CREATE INDEX 语句中指定了 ALLOW REVERSE SCANS 参数时可以按逆向方向搜索值，但是，执行指定索引次序的扫描比执行逆向扫描稍微更快一些。
- 要节省索引维护成本和空间：
  ——避免创建的索引是这些列上其他索引键的部分键。例如，如果列 a，b 和 c 上有索引，则列 a 和 b 上的第二个索引一般用处不大
  ——不在所有列上任意创建索引。不必要的索引不仅占用空间，而且会导致大量准备时间。当使用具有动态编程连接枚举的优化级别时，对于复杂的查询这特别重要。
  ——使用下列一般规则来确定将为表定义的索引的典型数目。此数目根据数据库的主要使用来确定：
    对于在线事务处理(OLTP)环境，创建 2 个或 3 个索引
    对于只读查询环境，可以创建 5 个以上索引
    对于混合查询和在线事务处理环境，可以创建 2 到 5 个索引
- 要改进对父表执行的删除和更新操作的性能，在外键上创建索引。
- 对于快速排序操作，在频繁用于排序数据的列上创建索引。
- 要改进多列索引的连接性能，如果第一个键列有多项选择，则使用最常用的“=”(等值连接)谓词指定的那一列，或使用如第一个键那样具有最多不同值的那些列。
- 要帮助新插入的行根据索引进行群集并避免页分割，定义一个集群索引。集群索引应显著减少重组表的需要。

  当定义表时使用 PCTREE 关键字来指定页上应该留下多少可用空间，才能允许将

插入行适当地放在页上。也可以指定 LOAD 命令的 pagefreespace MODIFIED BY 子句。

- 要启用联机索引整理碎片，创建索引时使用 MINPCTUSED 选项。MINPCTUSED 指定索引叶子页中最小使用空间量的阀值并启用联机索引整理碎片。如果这些删除实际上是从索引页除去键，则可以在键删除期间以性能损失为代价而减少重组的需要。

在下列情况下，考虑创建索引：

- 在最频繁处理的查询和事务的 WHERE 子句中所使用的那些列上创建索引。

例如以下的 WHERE 子句

```
WHERE  WORKDEPT=A01 OR WORKDEPT=E21
```

通常将会从 WOPKDEPT 上的索引获益，除非 WORKDEPT 列包含许多重复值。

- 在按查询所需要的顺序对行排序的一列或多列上创建索引。不仅在 ORDER BY 子句中，而且其他功能，如 DISTINCT 和 GROUP BY 子句也都需要排序。

  例如以下示例使用 DISTINCT 子句：

```
SELECT DISTINCT WORKDEPT     FROM EMPLOYEE
```

数据库管理器可以使用 WORKDEPT 上定义为升序或降序的索引来消除重复值。此时一个索引也可用于 GROUP BY 子句中，以将值分组，如下所示：

```
 SELECT WORKDEPT, AVERAGE(SALARY) FROM EMPLOYEE GROUP BY WORKDEPT
```

- 使用复合键创建索引，该键命名语句中引用的每个列。当用此方式指定索引时，可以从纯索引检索数据，这比存取表更有效。

  例如，考虑下列 SQL 语句：

```
SELECT LASTNAME  FROM EMPLOYEE  WHERE WORKDEPT IN('A00', 'D11', 'D21')
```

如果为 EMPLOYEE 表的 WORKDEPT 和 LASTNAME 列定义了索引，那么通过扫描索引而不是扫描整个表可能会更有效地处理该语句。注意，因为该谓词基于 WORKDEPT，因此此列应是该索引的第一列。

- 使用 INCLUDE 列创建索引可改善表上索引的使用。使用上述示例，可将唯一索引定义为：

```
CREATE UNIQUE INDEX x ON employee(workdept) INCLUDE(lastname)
```

指定 lastname 为 INCLUDE 列而不是索引键的一部分，意味着 lastame 只存储在索引的叶子页上。

5.4 使用序列提高性能

序列是一个数据库对象，它允许自动生成值，例如支票号、流水号、订单号等。序列特别适合于生成唯一键值这一任务。应用程序可以使用序列来避免用于跟踪数字的列值所引起的可能的并行性和性能问题。与在数据库外部创建的数字相比，序列的优点在于数据库服务器会跟踪生成的数字。崩溃和重新启动不会导致生成重复的数字。

5.4.1 应用程序性能和序列

与其他方法相比，使用序列来生成值通常会提高应用程序的性能，这一点与标识列相同。序列的替代方法是创建存储当前值的单列表并使用触发器或在应用程序控制下递增值。但是，在一个分布式环境中，如果应用程序当前访问单列表，那么强制对该表进行序列化访问所需的锁定可能会严重影响性能。

使用序列可以避免与单列表方法关联的锁定问题，并且可以将序列值高速缓存在内存中以减少响应时间。为了让使用序列的应用程序的性能最高，请确保序列高速缓存适当数量的序列值。CREATE SEQUENCE 和 ALTER SEQUENCE 语句的 CACHE 子句指定数据库管理器生成并存储在内存中的最大数目的序列值。

如果序列必须按顺序生成值，并且不会由于系统故障或数据库取消激活而在该顺序中引入间隔，请在 CREATE SEQUENCE 语句中使用 ORDER 和 NO CACHE 子句。NO CACHE 子句保证生成的值中没有间隔，但会使应用程序性能降低一些，因为它每次生成新值时，都会强制将序列写入数据库日志。请注意，由于事务回滚并且未真正使用它们请求的该序列值，所以仍然会出现间隔。

生成的序号具有下列属性：

- 值可以是小数位为零的任何精确数字数据类型。这样的数据类型包括：SMALL INT、BIG INT、INTEGER 和 DECIMAL。
- 连续值之间可以有任何指定的整数增量。默认递增值是 1。
- 计数器值是可恢复的。当需要恢复时，从日志中重建计数器值。
- 可以高速缓存值以改善性能。在高速缓存中预分配并存储值，可以在为序列生成值时减少对日志的同步 I/O。在系统出现故障时，将认为尚未使用的所有高速缓存值已丢失。为 CACHE 指定的值是可能丢失的序列值的最大数目。

有两种表达式可与序列一起使用：

- NEXT VALUE 表达式：对指定序列返回下一个值。当 NEXT VALUE 表达式指定序列的名称时，将生成一个新的序号。但是，如果一个查询中有多个 NEXT VALUE 表达式的实例指定同一序列名，那么对于结果的每一行，序列计数器仅递增一次，且 NEXT VALUE 的所有实例对结果的每一行返回同一个值。
- PREVIOUS VALUE 表达式：对于当前应用程序进程中的先前语句，该表达式对指定序列返回最新生成的值。也就是说，对于任何给定连接，PREVIOUS VALUE 将保持不变，即使另一个连接调用 NEXT VALUE 也是如此。

5.4.2 设计序列原则

设计序列时，需要考虑标识列与序列之间的差别，以及哪个更适合您的环境。如果决定使用序列，那么您需要熟悉可用的选项和参数。

可以通过调整序列的行为来满足应用程序要求。在发出 CREATE SEQUENCE 语句以创建新序列或对现有序列发出 ALTER SEQUENCE 语句时，可以更改序列的属性。除了容易设计和创建外，序列还具有其他各种选项，它们允许您更灵活地生成值：

- 从各种数据类型(SMALL INT、INTEGER、BIGINT 或 DECIMAL)中选择
- 更改起始值(START WITH)
- 更改序列增量，包括指定不断增大或不断减小的值(INCREMENT BY)
- 设置最小值和最大值，即序号的起始值和结束值(MINVALUE/MAXVALUE)
- 允许回绕值以便序列可以再次重新开始，或者禁止循环(CYCLE/NO CYCLE)
- 允许高速缓存序列值以提高性能，或者禁止高速缓存(CACHE/NO CACHE)

即使在生成序列后，这些值中的许多值也可以改变。例如，您可能要根据星期几来设置另一个起始值。使用序列的另一个实际示例是生成和处理银行支票。银行支票号序列非常重要，如果一组序号丢失或损坏，那么将会产生严重后果。

为了提高性能，还应该了解并使用 CACHE 选项。此选项告知数据库管理器在系统生成多少个序列值后，才返回到目录以生成另一组序列。如果未指定 CACHE 值，那么默认值为 20。以默认值为例，在请求第一个序列值时，数据库管理器将自动在内存中生成 20 个连续值(1，2，……，20)。每次需要新的序号时，就会使用此内存高速缓存值来返回下一个值。用完此高速缓存值后，数据库管理器将生成下一组 20 个值(21，22，……，40)。

通过实现序号的高速缓存，数据库管理器不必始终转至目录表来获取下一个值。这将减少与检索序号相关的开销，但在系统出现故障或关闭时，它可能还会导致序列中出现间隔。例如，如果您决定将序列高速缓存设置为 100，那么数据库管理器将高速缓存 100 个

这样的数字值，并且还将设置系统目录以表明下一个序列值应从 201 开始。在数据库关闭时，下一组序号将从 201 开始。如果未使用生成的从 101 到 200 的数字，那么这些数字将从序列集中丢失。如果您的应用程序无法容忍生成的值中出现间隔，那么需要将高速缓存值设置为 NO CACHE，虽然这样会产生较高的系统开销。

注意：

只有在不需要唯一数字或者可以保证在序列循环后不再使用较旧的序列值时，才使用 CYCLE。

例如，如果要创建一个名为 xinzhuang_values 的序列，其最小值为 0、最大值为 1000、NEXT VALUE 表达式使值递增 1，并且在达到最大值时返回到其最小值，那么请发出以下语句：

```
CREATE SEQUENCE xinzhuang_values  START WITH 0
  INCREMENT BY 1 MAXVALUE 1000  CYCLE
```

5.4.3 序列维护

1. 创建序列

要创建序列，请使用 CREATE SEQUENCE 语句。与标识列属性不同，未使序列与特定表列相关，也未将它绑定至唯一表列，只是仅可通过该表列访问。

在可使用 NEXT VALUE 或 PREVIOUS VALUE 表达式的位置有几个限制。可以创建或改变序列，以便序列以下列其中一种方式来生成值：

- 单调地递增或递减(按常量更改)且没有限制
- 单调地递增或递减至用户定义的限制并停止
- 单调地递增或递减至用户定义的限制并循环回至起点，然后重新开始

注意：

在恢复使用序列的数据库时请务必小心。

对于在数据库外部使用的序列值(例如，用于银行支票的序号)，如果将数据库恢复至数据库失败前的一个时间点，那么可能会导致对某些序列生成重复值。要避免可能的重复值，就不应该将在数据库外部使用序列值的数据库恢复至前一时间点。

例如，要使用所有选项的默认值来创建一个名为 order_seq 的序列，在应用程序中或通过使用动态 SQL 语句来发出以下语句：

```
CREATE SEQUENCE order_seq
```

此序列从 1 开始，并以 1 为增量增加，且没有上限。

下面这个语句可以表示处理从 101 开始至 200 的一组银行支票。第一个顺序应该是从 1 到 100。序列从 101 开始并以 1 为增量增加，其上限为 200。指定 NOCYCLE 以便不会产生重复的支票号。与 CACHE 参数关联的数指定了序列值的最大数目，数据库管理器预分配此数目并将它保存在内存中。

```
CREATE SEQUENCE order_seq   START WITH 101 INCREMENT BY 1
      MAXVALUE 200     NOCYCLE     CACHE 25
```

生成顺序值

生成顺序值是一个常见的数据库应用程序开发问题。解决该问题的最好方法是在 SQL 中使用序列和序列表达式。每个序列是只能由序列表达式访问的唯一已命名数据库对象。

有两个序列表达式：PREVIOUS VALUE 和 NEXT VALUE。PREVIOUS VALUE 表达式对指定的序列返回应用程序进程中最新生成的值。与 PREVIOUS VALUE 表达式出现在同一语句中的任何 NEXT VALUE 表达式不会影响该语句中的 PREVIOUS VALUE 表达式生成的值。NEXT VALUE 序列表达式使序列值递增并返回序列的新值。

要创建序列，请发出 CREATE SEQUENCE 语句。例如，要使用默认属性创建一个名为 xinzhuang_values 的序列，请发出以下语句：

```
CREATE SEQUENCE xinzhuang_values
```

要在序列的应用程序会话中生成第一个值，请使用 NEXT VALUE 表达式的 VALUES 语句：

```
VALUES NEXT VALUE FOR xinzhuang values
  1
-----------
  1
1 record(s) selected.
```

要将列值更新为序列的下一个值，请在 UPDATE 语句中包括 NEXT VALUE 表达式，如下所示：

```
UPDATE staff  SET id = NEXT VALUE FOR xinzhuang_values  WHERE id = 350
```

要使用序列的下一个值将新行插入到表中，请在 INSERT 语句中包括 NEXT VALUE 表达式，如下所示：

```
INSERT INTO staff(id, name, dept, job) VALUES(NEXT VALUE FOR xinzhuang_values,
```

```
'Kandil', 51, 'Mgr')
```

创建序列示例

编写的许多应用程序需要使用序号来跟踪发票号、客户编号、订单号、流水号，以及每次需要新项时其编号就会增大 1 的其他对象。通过使用标识列，数据库管理器可以使表中的值自动递增。虽然这项技术对于单独的表来说效果不错，但它可能不是生成多个表中需要使用的唯一值的最方便方法。

序列对象允许您创建在程序员控制下递增并且可以在许多表中使用的值。以下示例说明了如何为客户编号创建数据类型为 INTEGER 的序号：

```
CREATE SEQUENCE customer_no AS INTEGER
```

默认情况下，序号从 1 开始并且每次递增 1，其数据类型为 INTEGER。应用程序需要使用 NEXT VALUE 表达式来获取序列中的下一个值。此表达式生成序列的下一个值，然后可以将该值用于后续 SQL 语句：

```
VALUES NEXT VALUE FOR customer_no
```

程序员可以在 INSERT 语句中使用 VALUES 函数，而不是使用此函数生成下一个数字。例如，如果 Customer 表的第一列包含客户编号，那么可以按如下所示编写 INSERT 语句：

```
INSERT INTO customers VALUES    (NEXT VALUE FOR customer_no, 'comment', ...)
```

如果需要对插入到其他表中的操作使用序号，那么可以使用 PREVIOUS VALUE 表达式来检索先前生成的值。例如，如果需要将刚刚创建的客户编号用于后续发票记录，那么 SQL 应包括 PREVIOUS VALUE 表达式：

```
INSERT INTO invoices    (34,PREVIOUS VALUE FOR customer_no, 234.44, ...)
```

PREVIOUS VALUE 表达式可以在应用程序内多次使用，并且它仅返回该应用程序生成的最后一个值。后续事务可能已将序列递增至另一个值，但您看到的始终是生成的最后一个值。

2. 修改序列

使用 ALTER SEQUENCE 语句修改现有序列的属性。

可以修改的序列属性包括：

- 更改将来值之间的增量
- 建立新的最小值或最大值

- 更改高速缓存序号的数目
- 更改序列是否将循环
- 更改是否必须按请求顺序生成序号
- 重新启动序列

有两种任务不是序列创建的一部分。它们是：

- RESTART：将序列复位为隐式或显式指定的值，该值是在创建序列时作为起始值指定的。
- RESTART WITH <numeric-constant>：将序列复位为准确的数字常数值。数字常数可以是任何正数值或负数值，而且任何小数点右边不带有非零数字。

在重新启动序列或更改为 CYCLE 之后，可能会生成重复序号。ALTER SEQUENCE 语句仅影响将来的序号。

不能更改序列的数据类型。而是必须删除当前序列，然后创建新序列，指定新的数据类型。

在改变序列时，会丢失数据库管理器未使用的所有高速缓存序列值。

3. 查看序列定义

使用包含 PREVIOUS VALUE 选项的 VALUES 语句来查看与序列相关的参考信息或查看序列本身。

要显示序列的当前值，请发出使用 PREVIOUS VALUE 表达式的 VALUES 语句：

```
VALUES PREVIOUS VALUE FOR xinzhuang_values
            1
-----------
            1
            1record(s) selected.
```

可以重复检索序列的当前值，并且在发出 NEXT VALUE 表达式之前，该序列返回的值不变。在以下示例中，PREVIOUS VALUE 表达式返回值为 1，直到当前连接中的 NEXT VALUE 表达式使序列的值递增为止：

```
VALUES PREVIOUS VALUE FOR xinzhuang values
           1
  -----------
           1
           1record(s) selected.
VALUES NEXT VALUE FOR xinzhuang_values
```

```
          1
    -----------
          2
          1record(s) selected.
```

即使另一个连接在同时使用序列值也是如此。

4. 删除序列

要删除序列，请使用 DROP 语句。

在删除序列时，语句的授权标识必须具有 SYSADM 或 DBADM 权限。

可以通过使用下列命令来删除特定序列：

```
DROP SEQUENCE <sequence_name>
```

其中<sequence_name>是要删除的序列名，它包括隐式或显式模式名以正确标识现有的序列。不能使用 DROP SEQUENCE 语句删除系统为 IDENTITY 列创建的序列。一旦删除序列，那么也会删除对该序列的所有特权。

5.4.4 比较序列与标识列

虽然对于 DB2 应用程序来说，序列和标识列用途相似，但它们之间存在一个重要差别。标识列使用装入实用程序自动生成单个表中的列值。序列根据请求使用 CREATE SEQUENCE 语句生成可在任何 SQL 语句中使用的顺序值。

1. 确定何时使用标识列或序列

虽然标识列和序列之间存在相似之处，但是也存在差别。在设计数据库和应用程序时可以使用各自的特征。

根据您的数据库设计和使用数据库的应用程序，下列特征将帮助您确定何时使用标识列以及何时使用序列。

标识列特征

- 标识列自动为单个表生成值
- 当将标识列定义为 GENERATED ALWAYS 时，始终由数据库管理器生成所用的值。在修改表的内容期间，不允许应用程序来提供它们自己的值
- 在插入行后，通过使用 IDENTITY_VAL_LOCAL()函数或 SELECT FROM INSERT 语句从插入中重新选择标识列，可以检索生成的标识值

- 装入实用程序可以生成标识值

序列特征

- 未使序列与任何一个表相关
- 序列生成可在任何 SQL 或 XQuery 语句中使用的顺序值

由于任何应用程序都可以使用序列，所以有两种表达式可用来控制如何检索指定序列中的下一个值和正在执行的语句之前生成的值。对于当前会话中的先前语句，PREVIOUS VALUE 表达式对指定序列返回最新生成的值。NEXT VALUE 表达式对指定序列返回下一个值。使用这些表达式允许在几个表内的几个 SQL 和 XQuery 语句中使用相同值。

2. 标识列

允许数据库管理器自动为添加至表的每一行生成唯一数字值。如果您正在创建一个表并且知道需要唯一标识将添加至该表的每一行，那么可通过 CREATE TABLE 语句向该表添加一个标识列。

```
CREATE TABLE <table name>
      (<column name 1> INT,
       <column name 2>, DOUBLE,
       <column name 3> INT NOT NULL GENERATED ALWAYS AS IDENTITY
        (START WITH <value 1>, INCREMENT BY <value 2>))
```

在上面示例中，第三列标识标识列。可以定义的其中一个属性是，在添加行时用来唯一定义每一行的列中使用的值。INCREMENT BY 子句后面的值显示对于添加至该表的每一行来说，标识列内容的后续值的增量。

创建标识属性后，可以使用 ALTER TABLE 语句更改或除去这些属性。还可以使用 ALTER TABLE 语句在其他列中添加标识属性。

3. 序列

允许自动生成值。序列特别适合于生成唯一键值这一任务。应用程序可以使用序列，来避免通过其他方法生成唯一计数器所引起的可能的并行性和性能问题。与标识列不同，未使序列与特定表列相关，也未将它绑定至唯一表列，只是仅可通过该表列访问。

可以创建序列并在以后改变它，以便它通过无限递增或递减值来生成值；或者递增或递减至用户定义的限制，然后停止；或者递增或递减至用户定义的限制，然后循环回至起点并重新开始。序列仅在单分区数据库中受支持。

以下显示如何创建一个名为 orderseq 的序列：

```
CREATE SEQUENCE orderseq          START WITH1 INCREMENT BY 1
              NOMAXVALUE    NOCYCLE     CACHE 50
```

在上面示例中，序列从 1 开始，并以 1 为增量增加，且没有上限。由于没有指定上限，所以没有理由循环回至起点并从 1 重新开始。CACHE 参数指定了数据库管理器预分配并保存在内存中的序列值的最大数目。

5.5 视图

5.5.1 视图类型

视图可从一个或多个表、昵称或视图中派生，且可以在检索数据时与表互换使用。当对视图中显示的数据进行更改时，该数据会在表中自行更改。在创建视图之前，视图所基于的表、昵称或视图必须已经存在。

可以创建视图来限制对敏感数据的访问，同时又允许对其他数据进行更多的一般访问。

当插入到一个视图中，而其视图定义中的选择列表直接或间接地包括一个表的标识列的名称时，标识列的同一规则也适用，就像 INSERT 语句直接引用该表的标识列一样。

除按上述方式使用视图外，视图还可以用于：

- 改变表而不影响应用程序。这可通过创建一个基于基础表的视图来完成。使用基础表的应用程序不会因新视图的创建而受影响。新的应用程序可将创建的视图用于与那些使用基础表的应用程序不同的目的。
- 对一列中的值求和，选择最大值，或计算平均值。
- 访问一个或多个数据源中的信息。可在 CREATE VIEW 语句内引用昵称，并可创建多个位置/全局视图(该视图可以连接位于不同系统中多个数据源的信息)。当使用标准的 CREATE VIEW 语法创建一个引用昵称的视图时，将看到一个警告，警告目前视图用户的认证标识，而不是视图创建者的认证标识将，用于访问数据源处的基本对象。使用 FEDERATED 关键字可以阻止此警告。

视图是高效率的数据呈示方法(无需维护数据)。视图不是实际的表，不需要永久存储器。“虚拟表”是即创建即使用的。

视图提供了另一种查看一个或多个表中的数据的方法。视图和表一样具有列和行。可以像使用表一样将所有视图用于数据检索。是否可以在插入、更新或删除操作中使用视图取决于它的定义。

视图可以包括它所基于的表中的所有或某些列或行。例如，可以在视图中连接一个部门表和一个职员表，以便可以列示特定部门中的所有职员。图 5-23 显示了表与视图之间的关系。

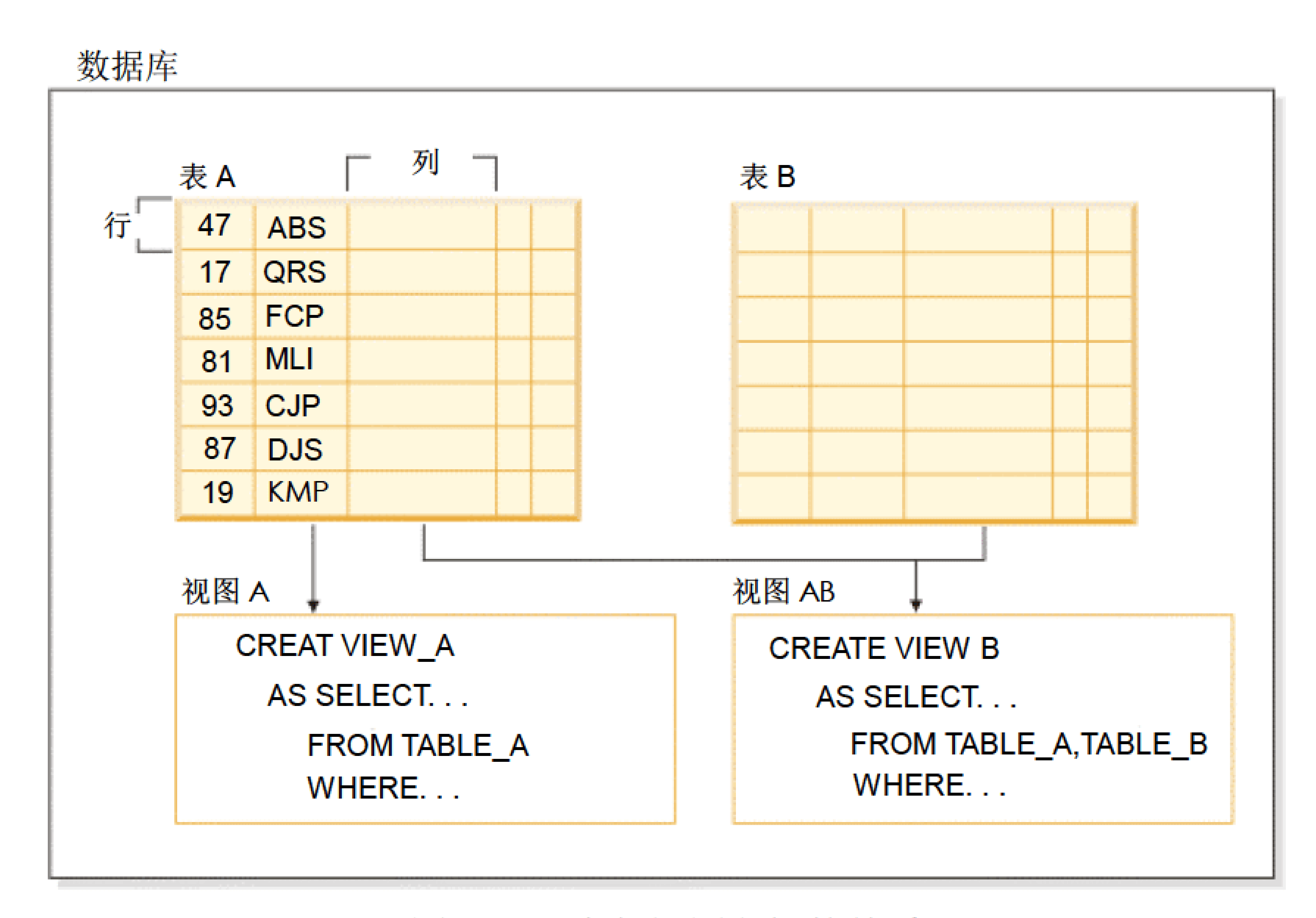


图 5-23　表与视图之间的关系

因为视图允许多个用户查看相同数据的不同表示，所以可以使用视图来控制对敏感数据的访问。例如，几个用户正在访问关于职员的数据表。经理可以看到关于他/她的职员的数据，但看不到关于其他部门中的职员的数据；招聘专员可以看到所有职员的聘用日期，但看不到他们的薪水；财务人员可以看到薪水，但看不到聘用日期。这些用户中的每个用户都使用派生自表的视图。每个视图都显示为一个表，并且具有自己的名称。

当视图列直接派生自基本表的列时，该视图列将继承适用于该表列的所有约束。例如，如果视图包括其表的外键，那么使用该视图的插入和更新操作时应遵守与该表相同的引用约束。此外，如果视图的表是一个父表，那么使用该视图的删除和更新操作时应遵守与对表执行删除和更新操作相同的规则。

视图有以下几种类型：

系统目录视图

数据库管理器维护一组表和视图，这些表和视图包含关于数据库管理器所控制的数据的信息。这些表和视图统称为系统目录。系统目录包含关于数据库对象(例如表、视图、索引、程序包和函数)的逻辑和物理结构的信息。它还包含统计信息。数据库管理器确保系统目录中的描述始终准确。

系统目录视图类似于任何其他数据库视图。可以使用 SQL 语句来查询系统目录视图中的数据。可以使用一组可更新的系统目录视图来修改系统目录中的某些值。关于系统目录视图的详细信息，请参见《深入解析 DB2-高级管理、内部体系结构和诊断案例》一书。

可删除视图

根据定义视图的方式，视图可以是可删除视图。可删除视图是可以对其成功发出 DELETE 语句的视图。

只有在遵循下列规则的情况下，一个视图才能被视为可删除视图：

- 外部全查询的每个 FROM 子句仅标识一个表(不带有 OUTER 子句)、可删除视图(不带有 OUTER 子句)、可删除的嵌套表表达式或可删除的公共表表达式。
- 数据库管理器应该能够使用视图定义来派生表中要删除的行。下列操作使视图变得不可删除：
  - ◊ 使用 GROUP BY 子句或列函数将多行分组为一行将导致原始行丢失并使得视图不可删除。
  - ◊ 同样，从 VALUES 派生行时，没有要删除行的表。视图也将不可删除。
- 外部全查询不使用 GROUP BY 或 HAVING 子句。
- 外部全查询的选择列表中不包括列函数。
- 外部全查询不使用集合操作(UNION、EXCEPT 或 INTERSECT)，但 UNION ALL 除外。
- UNION ALL 的操作数中的表不能是相同的表，并且每个操作数必须可删除。
- 外部全查询的选择列表不包括 DISTINCT。

一个视图必须符合上面列示的所有规则才能被视为可删除视图。例如，下列视图是可删除视图。它遵循可删除视图的所有规则。

```
CREATE VIEW deletable_view     (number, date, start, end)
    AS      SELECT number, date, start, end      FROM employee.summary
      WHERE date='01012007'
```

可插入视图

可插入视图允许您使用视图定义来插入行。如果对视图定义了用于插入操作的 INSTEAD OF 触发器，或者视图中的至少一列可更新(与用于更新的 INSTEAD OF 触发器无关)，并且视图的全查询不包括 UNION ALL，那么该视图是可插入视图。当且仅当给定

行正好满足一个基础表的检查约束时，才能将该行插入到视图中(包括 UNION ALL)。要插入到包含不可更新列的视图中，必须从列列表中省略这些列。

下面创建的视图是一个可插入视图。但是，在本示例中，尝试插入视图将失败。这是因为表中存在不接受空值的列。这些列中的某些列未出现在视图定义中。尝试使用视图来插入值时，数据库管理器会尝试将一个空值插入到 NOT NULL 列中，不允许执行此操作。

```
CREATE VIEW insertable_view    (number, name, quantity)
   AS   SELECT number, name, quantify FROM ace.supplies
```

可更新视图

可更新视图是一种特殊的可删除视图。如果可删除视图中的至少一列可更新，那么该可删除视图就变成了可更新视图。

当满足下列所有规则时，视图中的一列将可更新：

- 视图是可删除视图
- 列解析为表列(不使用解引用操作)并且未指定 READ ONLY 选项
- 如果视图的全查询包含 UNION ALL，那么 UNION ALL 的操作数的所有相应列具有完全匹配的数据类型(包括长度或精度和小数位)以及完全匹配的默认值。

以下示例使用无法更新的常量值。但是，视图是可删除视图并且该视图中至少一列可更新。因此，它是可更新视图。

```
    CREATE VIEW updatable view    (number, current date, current time,
temperature)
       AS     SELECT number, CURRENT DATE, CURRENT TIME, temperature)
       FROM weather.forecast    WHERE number = 300
```

只读视图

如果一个视图不可删除、更新或插入，那么该视图是只读视图。SYSCAT. VIEWS 目录视图中的 READONLY 列指示视图是只读(R)视图。

下面创建的视图不是可删除视图，因为它使用了 DISTINCT 子句并且 SQL 语句涉及多个表：

```
    CREATE VIEW read only view    (name, phone, address)
     AS SELECT DISTINCT viewname, viewphone, viewaddress
       FROM employee.history adam, employer.deptSALES   WHERE adam.id = sales.id
```

5.5.2 创建 with check option 视图

下面显示了样本 CREATE VIEW 语句。基础表 EMPLOYEE 具有 SALARY 列和 COMM 列。为了安全起见，仅根据 ID、NAME、DEPT、JOB 和 HIREDATE 列创建此视图。此外，对 DEPT 列的访问受限制。此定义仅显示属于 DEPTNO 为 10 的部门的职员信息。

```
CREATE VIEW EMP VIEW1     (EMPID, EMPNAME, DEPTNO, JOBTITLE, HIREDATE)
    AS SELECT ID, NAME, DEPT, JOB, HIREDATE FROM EMPLOYEE   WHERE DEPT=10;
```

在定义视图后，可以指定访问特权。由于只能访问表的受限视图，所以指定访问特权可以提供数据安全性。如上所示，视图可以包含 WHERE 子句以限制对某些行的访问，或者可以包含列子集以限制对某些数据列的访问。

视图中的列名不必与基本表的列名匹配。表名具有关联的模式，视图名也一样。

定义视图后，就可以在诸如 SELECT、INSERT、UPDATE 和 DELETE 之类的语句中使用它(但具有一些限制)。DBA 可以决定提供一组用户，他们对视图具有的特权级别比对表的特权级别要高。

创建带检查选项的视图

定义了 WITH CHECK OPTION 的视图将针对该视图的 SELECT 语句强制检查任何修改或插入的行。使用检查选项的视图也称为对称视图。例如，仅返回部门 10 中的职员的对称视图不允许插入其他部门中的职员。因此，此选项将确保数据库中修改的数据的完整性，并在 INSERT 或 UPDATE 操作期间违反条件时返回错误。

如果应用程序无法将需要的规则定义为表检查约束，或者规则不适用于数据的所有用法，那么可以使用另一种方法在应用程序逻辑中实施规则。可以考虑创建一个表视图，其对数据的条件包括在指定的 WHERE 子句和 WITH CHECK OPTION 子句中。此视图定义将数据检索限于对应用程序有效的行集。此外，如果您可以更新该视图，那么 WITH CHECK OPTION 子句将更新、插入和删除操作限于适用于应用程序的行。

不能对下列视图指定 WITH CHECK OPTION：

- 使用只读选项定义的视图(只读视图)
- 引用 NODENUMBER 或 PARTITION 函数、非确定性函数(例如，RAND)或使用外部操作的函数的视图

例 5-17 以下是一个使用 WITH CHECK OPTION 的视图定义的示例。需要此选项以确保始终检查条件。该视图确保 DEPT 始终为 10。这将限制 DEPT 列的输入值。使用视图来插入新值时，始终强制执行 WITH CHECK OPTION：

```
CREATE VIEW EMP_VIEW2(EMPNO, EMPNAME, DEPTNO, JOBTITLE, HIREDATE)
  AS SELECT ID, NAME, DEPT, JOB, HIREDATE FROM EMPLOYEE   WHERE DEPT=10
  WITH CHECK OPTION;
```

如果在 INSERT 语句中使用此视图，那么当 DEPTNO 列的值不是 10 时将拒绝行。一定要记住，在未指定 WITH CHECK OPTION 的情况下，在修改期间不会进行数据验证。

如果在 SELECT 语句中使用此视图，那么将会调用条件(WHERE 子句)并且生成的表仅包含匹配的数据行。也就是说，WITH CHECK OPTION 不影响 SELECT 语句的结果。

例 5-18　使用视图使程序和最终用户查询可以灵活地查看表数据。

下列 SQL 语句创建 EMPLOYEE 表的视图，它列示部门 A00 的所有职员及其姓名和电话号码：

```
CREATE VIEW EMP VIEW(DA00NAME, DA00NUM, PHONENO)
             AS SELECT LASTNAME, EMPNO, PHONENO FROM EMPLOYEE
             WHERE WORKDEPT = 'A00'         WITH CHECK OPTION
```

此语句的第一行对该视图命名并定义它的列。名称 EMP_VIEW 在 SYSCAT. TABLES 中的模式内必须是唯一的。尽管不包含数据，视图名看上去仍像一个表名。该视图将有称为 DA00NAME、DA00NUM 和 PHONENO 的 3 列，它们与 EMPLOYEE 表中的列 LASTNAME、EMPNO 和 PHONENO 相对应。列示的列名按一一对应的关系应用于 SELECT 语句的选择列表。如果不指定列名，那么视图使用与 SELECT 语句的结果表的列相同的名称。

第二行是描述要从数据库选择哪些值的 SELECT 语句。它可以包括子句：ALL、DISTINCT、FROM、WHERE、GROUP BY 和 HAVING。为视图提供列的数据对象的一个或多个名称必须跟在 FROM 子句后面。

5.5.3　视图维护

有些情况下，视图会变得不可用，不可用视图是指不能再用于 SQL 语句操作的视图。在下列情况下，视图可能变得不可用：

- 撤销了对基础表的特权
- 删除了表、别名或函数
- 当删除它们所从属的视图

下列步骤可以帮助 DBA 恢复不可用视图：

(1) 确定最初用于创建该视图的 SQL 语句。可以从 SYSCAT. VIEW 目录视图的 TEXT 列获取此信息。

(2) 将当前模式设置为 QUALIFIER 列的内容。

(3) 将函数路径设置为 FUNC_PATH 列的内容。

(4) 使用 CREATE VIEW 语句并使用相同的视图名和相同的定义来重新创建该视图。

(5) 使用 GRANT 语句重新授予先前在该视图上授予的所有特权(注意，在不可用视图上授予的所有特权都被撤销)。

如果不希望恢复不可用视图，可以使用 DROP VIEW 语句显式删除它。例如，以下示例显示如何删除名为 EMP_VIEW 的视图：

```
DROP VIEW EMP_VIEW
```

或者可以使用相同的名称和不同的定义来创建一个新视图。

5.6 表表达式

表表达式可用于代替定义视图，它主要用在需要使用它的结果表的查询语句中。视图的定义被存储在系统的编目表中，并且可由所有被授权的用户共享。表表达式是临时的，并且仅在包含它的 SQL 语句生存期间有效，它们不能被共享，但比视图具有更多的灵活性，可以减少对系统目录表的维护，提高性能。

表表达式包括以下内容：

5.6.1 嵌套的表表达式

嵌套的表表达式(Nested Table Expression)可以被认为是一个视图，只是它的定义嵌套(被直接地定义)在一个查询语句的 FROM 子句中。

下面的查询语句使用了一个嵌套的表表达式给出对如下询问的回答：对于教育级别高于 16 的那些雇员，总收入的平均值是多少？教育级别是什么？哪年雇佣的？

```
SELECT EDLEVEL, HIREYEAR, AVG(TOTAL PAY)
   FROM( SELECT EMPLNO,YEAR(HIREDATE) AS HIREYEAR,EDLEVEL,
                 SALARY+BONUS+COMM AS TOTAL PAY FROM EMPLOYEE
  WHERE EDLEVEL >16 ) AS PAY_LEVEL
    GROUP BY EDLEVEL,HIREYEAR
```

这个查询首先使用一个嵌套的表表达式从 HIREDATE 列中提取雇员的雇佣年号，以便在后边的 GROUP BY 语句中使用。在需要对表达式的结果进行分组时，表表达式是非常有用的。当打算进行类似的查询但使用不同的值作为对 EDLEVEL 的选择条件时，您可能觉得利用表表达式更为方便，而不想把它创建为一个视图，因为创建视图需要增加更多的维护工作。

5.6.2　公用表表达式

一个公用表表达式(Common Table Expretion)是一个被命名的结果表，它定义在一个完全选择的前面，并且以 WITH 关键字开始。赋给公用表表达式的标识符可在整个查询中的任何 FROM 子句中用作表名。每次对公用表表达式名字的重复引用，使用的结果表都相同。这点与嵌套的表表达式或视图是不同的，它们可能对每次引用给出具有不同结果的结果集。

下面的例子完成这样一个要求：找出公司中这样的雇员，他们的教育级别高于 16，但收入低于与他们同时雇佣且具有相同级别的雇员收入的平均值。在该语句之后详细分析这个查询语句的各部分。

```
[1] WITH
   PAYLEVEL AS
     (SELECT EMPNO,YEAR(HIREDATE) AS HIREYEAR,EDLEVEL,
             SALARY+BONUS+COMM AS TOTAL_PAY
             FROM EMPLOYEE
             WHERE EDLEVEL>16
      ),
[2] PAYBYED(EDUC_LEVEL,YEAR_OF_HIRE,AVG_TOTAL_PAY) AS
       (SELECT EDLEVEL,HIREYEAR,AVG(TOTAL PAY)
              FROM PAYLEVEL
              GROUP BY EDLEVEL,HIREYEAR
        )
[3] SELECT EMPNO,EDLEVEL,YEAR OF HIRE,TOTAL PAY,AVG TOTAL PAY
     FROM PAYLEVEL,PAYBYED
       WHERE EDLEVEL=EDUC LEVEL
             AND HIREYEAR=YEAR_OF_HIRE
             AND TOTAL_PAY<AVG_TOTAL_PAY
```

[1] 这是一个名为 PAYLEVEL 的公用表表达式，它的结果表含有雇员被雇佣的年号，该雇员的总收入以及他们的教育级别，并且仅包含了教育级别高于 16 的雇员的信息。PAYLEVEL 所使用的查询与前边例子中嵌套的表表达式相同。

[2] 这是一个名为 PAYBYED 的公用表表达式。PAYBYED(也即 PAY BY Education)使用前边的公用表表达式中创建的 PAYLEVEL 表确定教育级别、雇佣年号，以及每种教育级别中雇员的平均收入，并且以不同于选择列表中所用的列名来命名这个表所返回的列(如 EDUC_LEVEL)。这个公用表表达式产生一个名为 PAYBYED 的结果集，它与前边的例子中嵌套的表表达式所产生的结果集相同。

[3] 最后是给出所希望的实际查询结果。两个表(PAYLEVEL 和 PAYBYED)被连接起来，以找出这样的雇员：他们的总收入低于与其在同一时间雇佣的人的平均收入。注意，PAYLEVEL 在整个查询语句中使用了两次，而且在这两次查询的结果中所包含的行是相

同的。

5.7 触发器设计

触发器定义一组操作，在响应对指定表的插入、更新或删除操作时将执行这些操作。执行这样的 SQL 操作时，触发器被认为是已激活的。触发器是可选的，并且可使用 CREATE TRIGGER 语句定义。

可将触发器与引用约束和检查约束配合使用，以强制执行数据完整性规则。还可使用触发器来完成诸如更新其他表，自动生成或变换插入或更新的行的值，或者调用函数以执行如发出警报之类的任务。

对于定义或强制事务性业务规则而言，触发器是非常有用的机制，这些规则涉及数据的不同状态(例如，薪水增长不能超过 10%)。

使用触发器会设置逻辑以在数据库内强制使用业务规则。这表示应用程序不负责强制使用这些规则。对所有表强制使用的集中逻辑意味着更容易维护，因为在逻辑更改时，不需要更改应用程序。

使用触发器来执行下列操作：

- 验证输入数据
- 为新插入的行生成值
- 为交叉引用而从其他表中进行读取
- 为审计跟踪而向其他表写入

可使用触发器支持一般形式的完整性或业务规则。例如，在接受订单或更新摘要数据表之前，触发器可以检查客户的信用额度。触发器具有以下优点：

- 更快地开发应用程序：因为触发器存储在数据库中，所以不必编写触发器在每个应用程序中执行的操作。
- 更容易维护：一旦定义了一个触发器，那么当访问创建它所基于的表时，会自动调用该触发器。
- 业务规则的全局实现：如果业务策略改变，只需更改触发器而不必更改每个应用程序。

5.7.1 触发器的类型

DB2 支持下列类型的触发器：

前触发器

在更新或插入操作前运行。在实际修改数据库之前，可以修改要更新或插入的值。可以将在更新或插入操作前运行的触发器用于下列几种用途：

- 在数据库中实际更新或插入值之前检查或修改这些值。如果需要将用户看到的数据格式变换为某种内部数据库格式，那么这样做很有用。
- 运行用户定义的函数中编写的其他非数据库操作。

后触发器

在更新、插入或删除操作后运行。可以将在更新或插入操作后运行的触发器用于下列几种用途：

- 更新其他表中的数据。此功能对于保持数据之间的关系或保留审计跟踪信息很有用。
- 针对表或其他表中的数据检查。当引用完整性约束不适合或者表检查约束限制仅对当前表进行检查时，此功能对于确保数据完整性很有用。
- 运行用户定义的函数中编写的非数据库操作。在发出警报或更新数据库外的信息时，此功能很有用。

INSTEAD OF 触发器

描述如何对视图执行插入、更新和删除操作，这些视图太复杂，以致无法在本机支持这些操作。这种触发器允许应用程序将视图用作所有 SQL 操作(插入、删除、更新和选择)的唯一界面。

通常，INSTEAD OF 触发器包含视图主体中应用的逻辑的相反逻辑。例如，考虑一个用于解密其源表中的列的视图。此视图的 INSTEAD OF 触发器加密数据，然后将它插入到源表中，因此执行对称操作。

通过使用 INSTEAD OF 触发器，请求对视图执行的修改操作将替换为触发器逻辑，该逻辑代表视图执行操作。从应用程序的角度来看，这是透明地进行的，因为它看到所有操作都是对视图执行的。只允许将一个 INSTEAD OF 触发器用于给定主题视图的每种操作。

视图本身必须是隐式类型视图或解析为隐式类型视图的别名。此外，它不能是使用 WITH CHECK OPTION 定义的视图(对称视图)，或在其上直接或间接定义了对称视图的视图。

不同的触发器激活时间反映不同的触发器用途。基本上，前触发器是对数据库管理系统的约束子系统的扩展。因此，您通常使用它们来：

- 验证输入数据
- 自动生成新插入的行的值
- 为交叉引用而从其他表中进行读取

由于前触发器是在将触发器事件应用于数据库之前激活的，所以不使用它们来进一步修改数据库。因此，在检查完整性约束之前激活这些触发器。

相反，可以将后触发器视为每次特定事件发生时就在数据库中运行的应用程序逻辑的模块。作为应用程序的一部分，后触发器始终看到处于一致状态的数据库。请注意，它们在完整性约束验证后运行。因此，主要将它们用来执行应用程序也可以执行的操作。例如：

- 在数据库中继续执行修改操作
- 在数据库外执行操作，例如，用以支持警报。请注意，回滚触发器时不会回滚在数据库外执行的操作

比较而言，可以将 INSTEAD OF 触发器视为对定义该触发器的视图的反向操作的描述。例如，如果视图中的选择列表包含一个基于表的表达式，那么其 INSTEAD OF INSERT 触发器的主体中的 INSERT 语句将包含反向表达式。

因为前触发器、后触发器和 INSTEAD OF 触发器具有不同的性质，所以可以使用一组不同的 SQL 操作来定义前触发器、后触发器和 INSTEAD OF 触发器的触发操作。例如，前触发器中不允许更新操作，这是因为不能保证触发操作不会违反完整性约束。同样，前触发器、后触发器和 INSTEAD OF 触发器中支持不同的触发器粒度。

5.7.2 创建触发器示例

```
CREATE TRIGGER  1  trigger2
     2 AFTER  3 UPDATE OF  4 ON_HAND, MAX_STOCKED ON  5 PARTS
      6 REFERENCING NEW AS N OLD AS O
 7 FOR EACH ROW
 8 WHEN(N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
        9  BEGIN ATOMIC
      10  VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                                  N_ROW.ON_HAND,
                                  N_ROW.PARTNO));
         END@
```

1 触发器名称。

2 触发器触发的时间(BEFORE、AFTER 和 INSTEAD OF 子句)。

- 如果激活时间是 BEFORE，那么将在触发器事件执行之前对受影响的行集中的每行激活触发操作。因此，只有在前触发器完成每行的执行后才修改主题表。请注意，前触发器必须具有 FOR EACH ROW 粒度。
- 如果激活时间是 AFTER，那么将对受影响的行集中的每行或对语句激活触发操作，这取决于触发器粒度。此操作在触发器事件完成后，并且在数据库管理器检查触发器事件可能影响的所有约束(包括引用约束的操作)后发生。请注意，后触发器可以具有 FOR EACH ROW 或 FOR EACH STATEMENT 粒度。
- 如果激活时间是 INSTEAD OF，那么将对受影响的行集中的每行激活触发操作，而不是执行触发器事件。INSTEAD OF 触发器必须具有 FOR EACH ROW 粒度，并且主题表必须是视图。其他触发器均无法将视图用作主题表。

3 触发器事件(INSERT、DELETE 或 UPDATE)。

4 触发器影响的列。

5 主题表。触发器是建在该表上的，是对该表设计的业务逻辑。

6 使用转换变量访问触发器中的旧列值和新列值

在行级触发器中，用 REFERENCING NEW AS N OLD AS O(称为伪记录)来访问数据变更前后的值。但要注意，INSERT 语句插入一条新记录，所以没有:old 记录。而 DELETE 语句删除掉一条已经存在的记录，所以没有:new 记录。UPDATE 语句既有:old 记录，也有:new 记录，分别代表修改前后的记录。引用具体的某一列的值的方法是：

```
:old.字段名或:new.字段名
OLD AS correlation-name
```

指定一个相关名，它捕获行的原始状态(即在将触发操作应用于数据库之前)的中间结果集。

```
NEW AS correlation-name
```

指定一个相关名，它捕获在将触发操作应用于数据库时用于更新该数据库中的行的值的中间结果集。

7 触发粒度，有两种触发粒度：

```
FOR EACH ROW
```

它运行的次数与受影响的行集中的行数相同。如果需要引用触发操作所影响的特定行，请使用 FOR EACH ROW 粒度。

```
FOR EACH STATEMENT
```

它对整个触发器事件运行一次。如果受影响的行集为空(也就是说，当搜索式 UPDATE 或 DELETE 中的 WHERE 子句未限定任何行时)，那么 FOR EACH ROW 触发器不运行。但 FOR EACH STATEMENT 触发器仍运行一次。

8 定义触发器操作将触发的条件。

激活触发器将导致运行与该触发器关联的触发操作。每个触发器正好有一个触发操作，而该触发操作又有两个组件：可选的触发操作条件或 WHEN 子句以及触发语句集。

触发操作条件是触发操作的可选子句，它指定一个搜索条件，必须对该条件求值为 true 才能运行触发操作内的语句。如果省略 WHEN 子句，那么始终执行触发操作内的语句。

9 原子触发器所影响的 SQL 语句要么全部成功要么全部失败。

10 触发器的触发操作(触发语句)。

当创建原子触发器时，必须认真对待语句结束字符。默认情况下，命令行处理器将“;”当作是语句结束标记。您应该在脚本中手动编辑语句结束字符来创建原子触发器，以便使用一个非“;”的字符。例如，可以用另一个特殊字符(如“#”)替换“;”。还可以在 CREATE TRIGGER DDL 前面加上下列内容：

```
--#SET TERMINATOR @
```

要是更改了正在处理的 CLP 中的终止符，以下语法将复原它：

```
--#SET TERMINATOR
```

要通过命令行来创建触发器，请输入：

```
db2 -td@ -vf <script>
```

其中 delimitera 是备用语句结束字符，而<script>是使用新<delimiter>的已修改脚本。

5.7.3 触发器设计总结

灵活地使用触发器可以方便我们实现业务逻辑，从而避免应用程序编码。但是在一个高并发的表上尽量避免创建过多的触发器。

对于触发器，很多人认为不要使用，主要的原因是触发器不好控制和触发器影响性能。在这里我做一个总结，触发器作为大型数据库的组成部分，可以代替应用程序编码实现复杂的业务逻辑。它的一些功能是其他方法无法代替的，特别是它作为数据库约束的补充，所能进行的业务规则的约束，以及在跟踪和同步中所能起到的作用。

个人认为，触发器确实不好控制，这是因为：

- 触发器实现的是在表操作的同时，自动进行操作或者控制，在写触发器代码时必须考虑其特殊性。
- 必须限定触发器影响的记录，不能扩大。也就是说必须引用转换变量访问触发器中的旧列值和新列值：REFERENCING NEW AS N OLD AS O 中两个虚表限定了操作的范围。
- REFERENCING NEW AS N OLD AS O 中两个虚表限定的操作范围的作用域只限定触发器本身，其调用的存储过程是不能使用的。
- 写触发器必须考虑性能，因为其自动性。如果触发器性能不好，则可能拖垮一个系统。
- 必须考虑一次操作多条记录的情况，除非保证一次只操作一条记录，一般不能用变量暂存虚表的数据，否则就可能出现在批量操作情况下，触发器只处理最后一条记录的情况，这类错误可以说是触发器最常见的错误之一。
- 必须注意递归和嵌套触发器，因为触发器往往需要修改其他或者本表数据来实现其功能，这里的修改数据往往能再次触发触发器，这时就必须保证其嵌套或者递归过程不是无限的，不会造成死循环，DB2 对触发器的嵌套层数有最多 16 层的限制。
- 触发器不好调试，比起一般的存储过程，触发器是在修改数据过程中触发，调试难度更大。调试过程必须考虑所有情况，比如空表插入数据、已有数据插入新数据、一次插入多行数据、修改一条数据、修改多条数据、一次删除多条数据、影响 0 行的修改或者删除语句等等。

触发器不好控制，这就要求我们在决定是否使用触发器的时候需要非常谨慎。个人认为，对于约束功能，如果可以用其他数据库方法实现，比如唯一约束、外键约束、规则约束、不可空约束，那么就不要用触发器，触发器只用来完成这些方法实现不了的约束。对于可以用触发器完成的跟踪、同步功能，则要考虑是否必要，必要的时候才用。而对于特定业务需求实现的触发器，则需要与应用编程实现的优劣进行比较而做选择。

对于触发器影响性能的说法，我不是很同意，因为在触发器代码写得没有问题的前提下，触发器所做的工作并不是多余的，这些功能如果不在触发器中完成，就必须在存储过程中完成或者客户端用其他代码实现，所以在系统负担上基本是一样的。触发器唯一比其他实现方法多消耗的是触发器作为特殊的存储过程，必须有个被调用的过程。由于 DB2 有存储过程的预编译和过程缓存机制，所以这方面的开销不会很多。相反，在某些情况下，由于触发器是在记录更改过程中执行的，可以把某些服务器的负载分散到多个时间点去执行，所以一定程度上均衡了服务器负担，从而提高整体的性能。

5.8 本章小结

本章我们给大家讲解了如何设计和创建表、索引、视图、触发器等数据库对象。我们的关注点不应该放到如何创建的命令上，而是应该放到每种对象的应用场合上。理解它们的特性并尽量用最合理的数据库技术来实现我们的业务需求。其实还有一些概念我们本章没有讲解，例如：我们可以给表创建一个别名(alias)，在联合数据库中我们还可以给表创建一个昵称(nickname)，这些都不常用到。我只是希望给大家讲解最常用的概念、最实用的技术。

第 6 章

数据移动

在数据库的使用过程中，经常需要将一个数据库中的数据转移到另外的一个数据库中。我们常用的方法是利用某种类型的外部文件作为中介，将一个数据库中的某个(些)表中的数据导出到该外部文件中，然后把该文件中的数据导入到另外一个数据库中。

DB2 中实现以上功能的主要工具有 4 个：EXPORT、IMPORT、LOAD 和 DB2MOVE。其中，EXPORT 的功能是将表中的数据导出到外部文件中；而 IMPORT 和 LOAD 的功能是将外部文件中的数据导入到一个表中，IMPORT 和 LOAD 的功能类似，但在实现手段上有很大差异：DB2MOVE 工具可以将表导出到 IXF，然后使用 IMPORT CREATE 模式在不同的数据库中对其进行重建，以对一组表在不同的系统间进行复制。

本章主要讲解如下内容：

- 数据移动格式
- EXPORT
- IMPORT
- LOAD
- 数据移动注意事项
- DB2MOVE 和 DB2LOOK

6.1 数据移动格式

能够被 DB2 所支持用作数据移动的中间文件的格式有 4 种：非定界 ASCII 码文件(ASCII)、定界 ASCII 码文件(DEL ASCII)、WSF 文件、PC/IXF 文件，如图 6-1 所示。

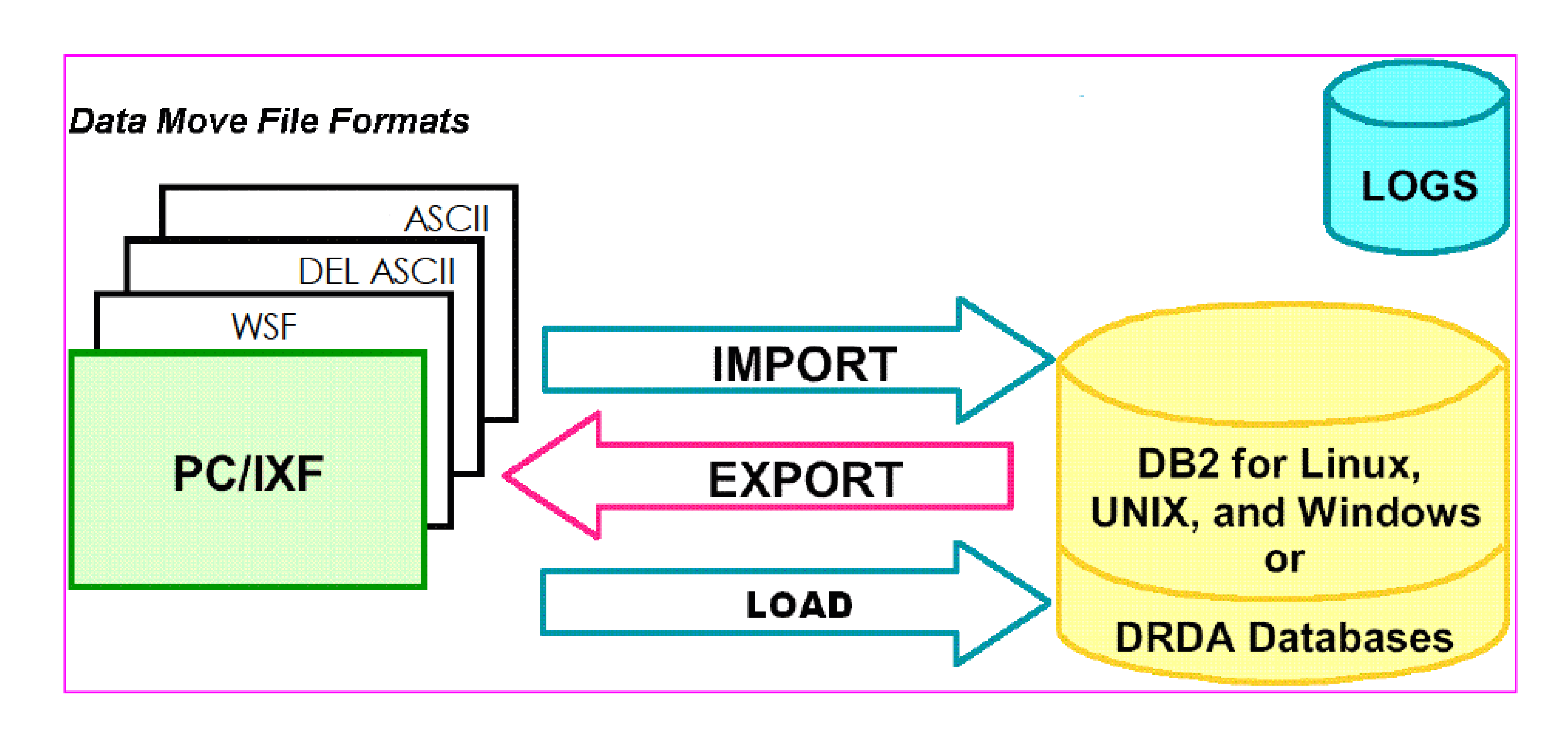


图 6-1　数据移动文件格式

6.1.1　定界 ASCII 文件格式

定界 ASCII 文件是带有行定界符和列定界符的顺序 ASCII 文件。每个 DEL 文件都是一个 ASCII 字符流，该字符流由先按行排序然后按列排序的单元值组成。数据流中的行由行定界符分隔，行中的列值由列定界符分隔。文件类型修饰符可用于修改这些定界符的默认值。

DEL 文件示例

以下是一个 DEL 文件示例。每一行都以换行符序列结尾(在 Windows 操作系统上，每一行都以回车符/换行符序列结尾)。

```
"Smith, Bob",4973,15.46
"Jones, Bill",12345,16.34
"Williams, Sam",452,193.78
```

6.1.2　非定界 ASCII 文件格式

非定界 ASCII 文件也是一个 ASCII 字符流。数据流中的行由行定界符分隔，而行中的每一列则通过起始和结束位置来定义。

非定界 ASCII 格式(对于 IMPORT 和 LOAD 实用程序来说称为 ASC)有两种变体：定长 ASC 和变长 ASC。对于定长 ASC 来说，所有记录都是定长的。对于变长 ASC 来说，记录由行定界符(始终是换行符)定界。在非定界 ASCII 中，术语非定界表示列值未由定界符分隔。

在导入或装入 ASC 数据时，如果指定了 reclen 文件类型修饰符，那么表示数据文件是定长 ASC 文件。如果未指定该修饰符，那么表示该数据文件是变长 ASC 文件。

ASC 文件示例

以下是一个 ASC 文件示例。每一行都以换行符序列结尾(在 Windows 操作系统上，每一行都以回车符/换行符序列结尾)。

```
Smith, Bob          4973    15.46
Jones, Suzanne     12345    16.34
Williams, Sam     452123   193.78
```

6.1.3 PC/IXF 文件格式

PC/IXF 文件由一些可变长的记录组成，包括标题记录、表记录、表中每个列的列描述符记录，以及表中每行的一个或多个数据记录。PC/IXF 文件记录由一些包含字符数据的字段组成。PC/IXF 文件格式是一种通用关系数据库交换格式，它支持很多关系数据类型，包括特定关系数据库产品可能不支持的某些类型。PC/IXF 文件格式保留了这一灵活性，例如，PC/IXF 体系结构同时支持单字节字符串(SBCS)和双字节字符串(DBCS)数据类型。它是一种非常通用的格式，被多种数据库管理系统所支持。可以用于在异构平台数据库间进行数据转移。

跨平台传输数据时，建议您使用 PC/IXF 文件格式。PC/IXF 是在跨平台转换数据时推荐使用的文件格式，因为它可以保留很多表属性，并且可以使用 DB2 数据移动工具以一种与平台无关的方式来处理数据。在 PC/IXF 文件中可以保留的表属性包括：主键、唯一约束、列信息(例如列名、数据类型和长度、可否为空属性以及标识属性)和索引信息。PC/IXF 文件中不能保留的表属性包括：引用和检查约束，以及部分列信息(例如默认值和所生成的列属性)。

6.1.4 工作表文件格式

WSF 是一种专有的二进制文件格式，它用于在 DB2、Lotus 1-2-3 和 Symphony 产品之间交换数据。WSF 文件不能被 LOAD 支持(要在导出操作期间创建与 WSF 格式相符的文件，可能会丢失一些数据)。如果工作中不使用 Lotus 1-2-3 和 Symphony 产品，您可以忽略这种数据格式。

6.1.5 游标

游标(CURSOR)是一个 SELECT 语句返回的结果集。这种格式仅限于 LOAD 使用，这种格式减少了中间文件的生成，可以提高 LOAD 的速度。

6.2 EXPORT

6.2.1 EXPORT 概述

下面我们首先来讲解 EXPORT 实用程序，它是几个实用程序中最简单的。EXPORT 实用程序会使用 SQL SELECT 语句或 XQuery 语句抽取数据，并将这些数据信息放到文件中。EXPORT 工具的本质是把一条 SQL 语句的结果集导出到一个文件中。EXPORT 面向的是 SQL 而不单纯的是表。您可使用输出文件移动数据以便将来执行导入或装入操作，或者将数据用于分析。

下列各项是基本导出操作所必需的：

- 要用于存储已导出数据的操作系统文件的路径和名称。
- 输入文件中的数据格式：EXPORT 支持对输出文件使用 IXF、WSF 和 DEL 数据格式。
- 指定要导出的数据：对于大部分导出操作，您需要提供 SELECT 语句指定需要进行检索以便导出的数据。

6.2.2 导出数据

使用 EXPORT 实用程序将数据从数据库导出至文件。该文件可使用若干外部文件格式中的一种。可以通过提供 SQL SELECT 语句来指定要导出的数据。

要想成功地调用 EXPORT 实用程序，必须拥有 SYSADM 或 DBADM 权限，或者拥有 EXPORT 命令中所访问的表或视图上的 CONTROL 或 SELECT 特权。在运行 EXPORT 实用程序之前，必须连接或能够隐式连接至要从中导出数据的数据库。因为实用程序会发出 COMMIT 语句，所以应在运行 EXPORT 实用程序之前发出 COMMIT 或 ROLLBACK 语句来完成所有事务并释放所有锁定。

EXPORT 实用程序是一个相对简单而且操作灵活的数据移动实用程序。可通过下列方法激活它：通过控制中心、在 CLP 中发出 EXPORT 命令或调用 ADMIN_CMD 存储过程。

1. 使用控制中心导出数据

要使用“导出表”图形界面导出数据：

(1) 在“控制中心”中，展开对象树，直到找到“表”或“视图”文件夹为止。

(2) 单击要使用的文件夹。任何现有的表或视图都将显示在窗口右边的窗格(内容窗格)中。

(3) 在内容窗格中，右击想要导出的表或视图，然后从弹出菜单中选择“导出”。“导出表”图形界面将打开如图 6-2 所示，您可以在图中定制一些导出选项。

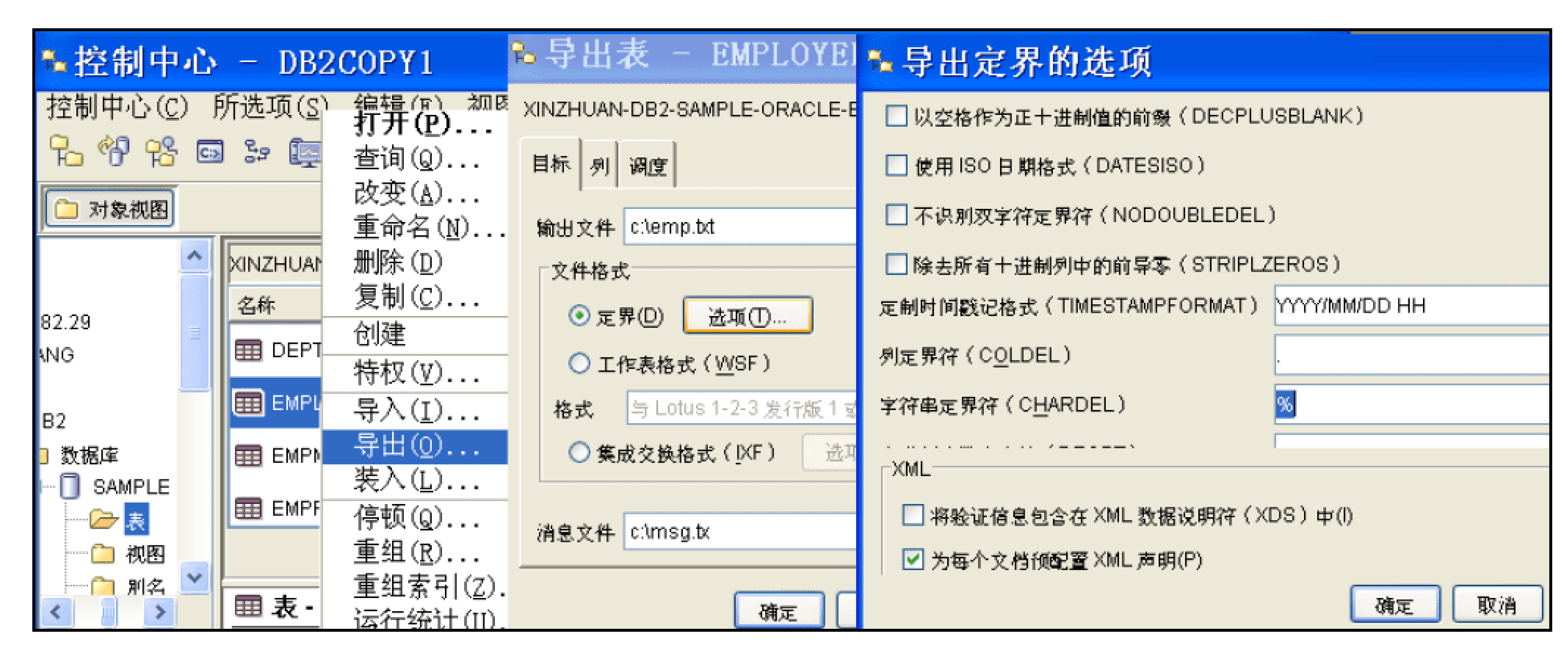


图 6-2　使用控制中心导出数据

2. EXPORT 命令

EXPORT 常用命令参数如图 6-3 所示。

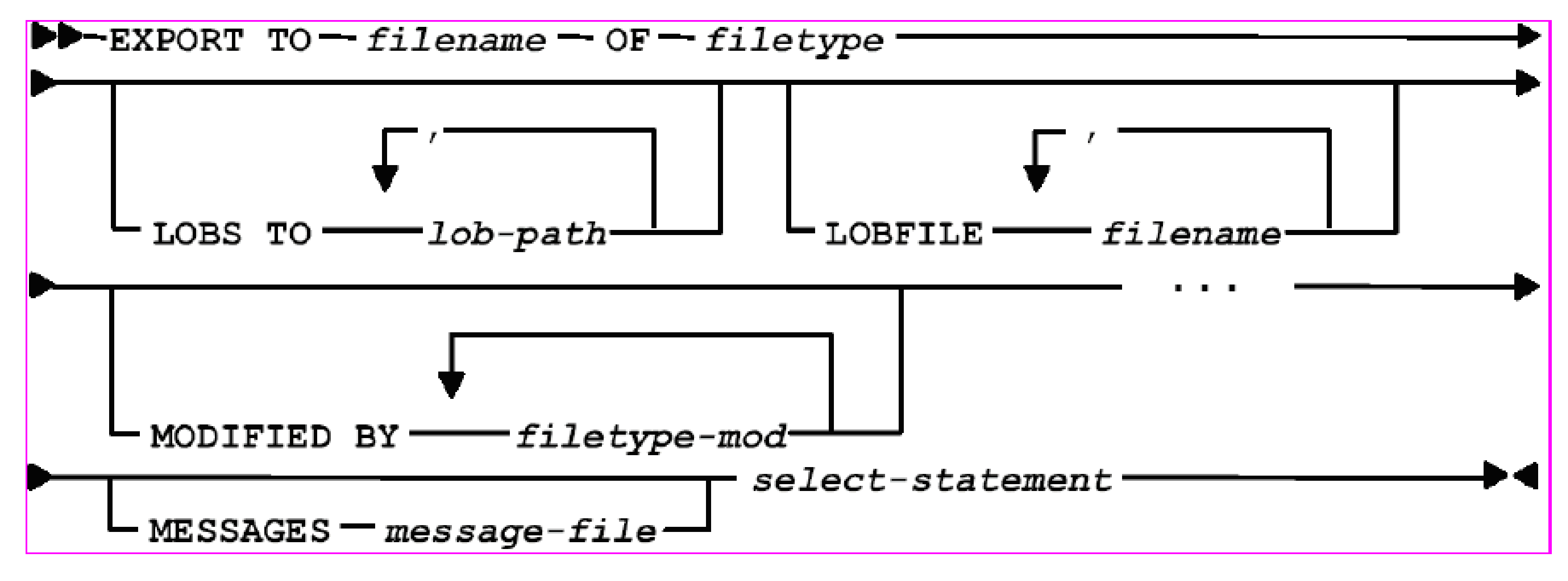


图 6-3　EXPORT 命令的常用命令参数

- TO filename

指定导出文件的名称。

- OF filetype

指定导出文件的类型。EXPORT 支持的导出格式是 DEL、WSF 和 IXF 格式。

- LOBFILE filename

指定 LOB 文件的一个或多个基文件名。如果表中有 LOB 类型数据要被导出，就指定一个或多个文件，用于存储 LOB 数据。

导出操作中创建 LOB 文件时，通过从这个列表向当前路径(lob-path)追加当前基名，然后追加一个 3 位序列号来构造文件名。例如，如果当前 LOB 路径为目录/u/foo/lob/path，并且当前 LOB 文件名为 bar，那么创建的 LOB 文件将为/u/foo/lob/path/bar.001、/u/foo/lob/path/bar.002 等等。

- LOBS TO lob-path

如果表中有 LOB 类型数据需要导出，就指定一个或多个目录文件，用于存储 LOB 数据。LOBFILE 参数指定的文件将存储在这个路径下。

- MESSAGES message-file

EXPORT 实用程序会将错误消息、警告消息和参考消息写至标准 ASCII 文本消息文件。对于 CLP 以外的所有接口，必须预先使用 MESSAGES 参数指定这些文件的名称。如果要使用 CLP 并且不指定消息文件，那么 EXPORT 实用程序会将消息写至标准输出。但是，您可能还想指定用于写入警告消息和错误消息的消息文件。为此，添加 MESSAGES 参数和消息文件名称。

- MODIFIED BY filetype-mod

指定一些额外的文件修饰符参数。文件类型修饰符提供了允许您更改数据、日期和时间戳记或代码页格式之类的许多选项，或者为输出文件指定特定的定界分隔符。

- METHOD N column-name

可指定要用于导出数据的不同列名。如果没有指定，将使用表中的相应列的名字。导出表仅仅支持 N 方法。

- select-statement

利用 SELECT 语句指定要导出的数据。

- XMLFILE、XML TO 和 XMLSAVESCHEMA

可从包括一个或多个 XML 数据类型列的表中导出数据。使用 XMLFILE、XML TO 和 XMLSAVESCHEMA 参数指定有关如何存储已导出文档的详细信息。

我们来看看一个简单的导出数据的例子。下面的命令将 SELECT 语句的结果导出到一个 DEL 格式的文件中。消息文件 msg.out 用于记录有用的信息和遇到的错误或警告：

```
    EXPORT TO myfile.del OF DEL   MESSAGES msg.out
    SELECT staff.name, staff.dept, org.location   FROM org, staff   WHERE
org.deptnumb = staff.dept;
```

其中，myfile.del 是要创建并导出的输出文件的名称，DEL 是文件格式，而 org 和 staff 是包含要导出的数据的表名。

3. 使用 ADMIN_CMD 存储过程

可以在命令行中直接调用 ADMIN_CMD 存储过程导出数据，如下所示：

```
    call sysproc.admin_cmd('export to /home/db2inst1/output/sales.del of del
messages /home/db2inst1/output/export.msg select * from sales')
```

6.2.3　导出数据示例

例 6-1　导出分界的文件。

```
EXPORT TO "D:\db2exp\employee.dat" OF DEL MESSAGES
"D:\db2exp\employee.log"      SELECT * FROM DB2ADMIN.EMPLOYEE;
```

例 6-2　将 XML 文档导出到单独的文件中。

在本示例中，我们看看如何导出 XML 文档。以下命令导出 *XEmployee* 表。每个 XML 文档放在单独的文件中，这些文件的基本名称是 *XEmployee*。使用以下命令之后，XEmployee.del 包含文档的列表(比如<XDS FIL='XEmployee.001.xml' />)，而包含数据的实际文档(比如 XEmployee.001.xml)导出到 XML TO 选项指定的目录中。在本示例中没有展示如何使用 XMLSAVESCHEMA 选项保存 XML 模式。

```
EXPORT TO "D:\db2XML\XEmployee.del" OF DEL XML TO "D:\db2XML\data"
XMLFILE "XEmployee" MODIFIED BY XMLINSEPFILES
MESSAGES "D:\db2XML\XEmployee.log"
SELECT * FROM "ALLAN WH THAM".XEMPLOYEE;
```

例 6-3　以下示例说明如何以 IXF 输出格式将 SAMPLE 数据库(用户必须连接至的数据库)的 STAFF 表中关于部门为 20 的职员的信息导出至 awards.ixf。

```
db2 export to awards.ixf of ixf messages msgs.txt select * from staff  where
dept = 20
```

例 6-4　以下示例说明如何将 LOB 导出到 DEL 文件：

```
 db2 export to myfile.del of del lobs to mylobs/
lobfile lobs1, lobs2 modified by lobsinfile     select * from emp_photo
```

例 6-5　以下示例说明如何将 LOB 导出到 DEL 文件，对可能无法装入到第一个目录中的文件指定第二个目录：

```
db2 export to myfile.del of del
lobs to /db2exp1/, /db2exp2/ modified by lobsinfile   select * from emp_photo
```

例 6-6　以下示例说明如何将数据导出到 DEL 文件，将单引号用作字符串定界符，分号用作列定界符，逗号用作小数点。在将数据导回数据库时应使用同一约束：

```
Db2 export to myfile.del of del
   modified by chardel '' coldel; decpt,  select * from staff
```

6.3 IMPORT

6.3.1 IMPORT 概述

IMPORT 实用程序会使用 SQL INSERT 语句向表、类型表或视图填充数据。如果目标表和目标视图中已包含数据，那么输入数据可替换(replace)现有数据，也可追加(insert)至现有数据。

6.3.2 导入数据

IMPORT 实用程序将具有受支持文件格式的外部文件中的数据插入到表、层次结构、视图或昵称中。

为了使用 DB2 IMPORT，必须获得正确的权限和特权，否则就无法顺利地执行导入。表 6-1 列出了将文件导入数据库所需的权限和特权：

表 6-1 DB2 IMPORT 所需的权限和特权

操　作	权　限	特　权	注　释
创建新的表	SYDADM/DBADM	CREATETAB	DB2 IMPORT 允许在导入期间动态地创建新表。这只能应用于表
在现有的表中插入数据	SYDADM/DBADM	CONTROL、INSERT 和 SELECT	可应用于表和视图
替换现有表中的数据	SYDADM/DBADM	CONTROL/(INSERT、SELECT、UPDATE 和 DELETE)	可应用于视图
在现有的表中追加数据	SYDADM/DBADM	SELECT 和 INSERT	可应用于视图

在调用 IMPORT 实用程序前，必须连接至(或者能够隐式连接至)要导入数据的数据库。由于实用程序将发出 COMMIT 或 ROLLBACK 语句，所以在调用导入之前，应该通过发出 COMMIT 或 ROLLBACK 操作以完成所有事务并释放所有锁定。

下列各项是基本导入操作所必需的：

- 输入文件的路径和名称

● 目标表或视图的名称或别名
● 输入文件中的数据格式
● 用于导入数据的方法

与 EXPORT 一样，IMPORT 是相对简单的数据移动实用程序。可通过下列方法激活它：通过控制中心、发出 CLP 命令、调用 ADMIN_CMD 存储过程或通过用户应用程序。

1. 使用控制中心导入数据

(1) 在“控制中心”中，展开对象树，直到找到“表”文件夹为止。

(2) 单击“表”文件夹。所有现有表都会显示在窗口右边的窗格(内容窗格)中。

(3) 在内容窗格中右击想要导入的表，然后从弹出菜单中选择“导入”。这就打开了“导入表”图形界面，如图 6-4 所示，您可以在图形界面中定制 IMPORT 选项。

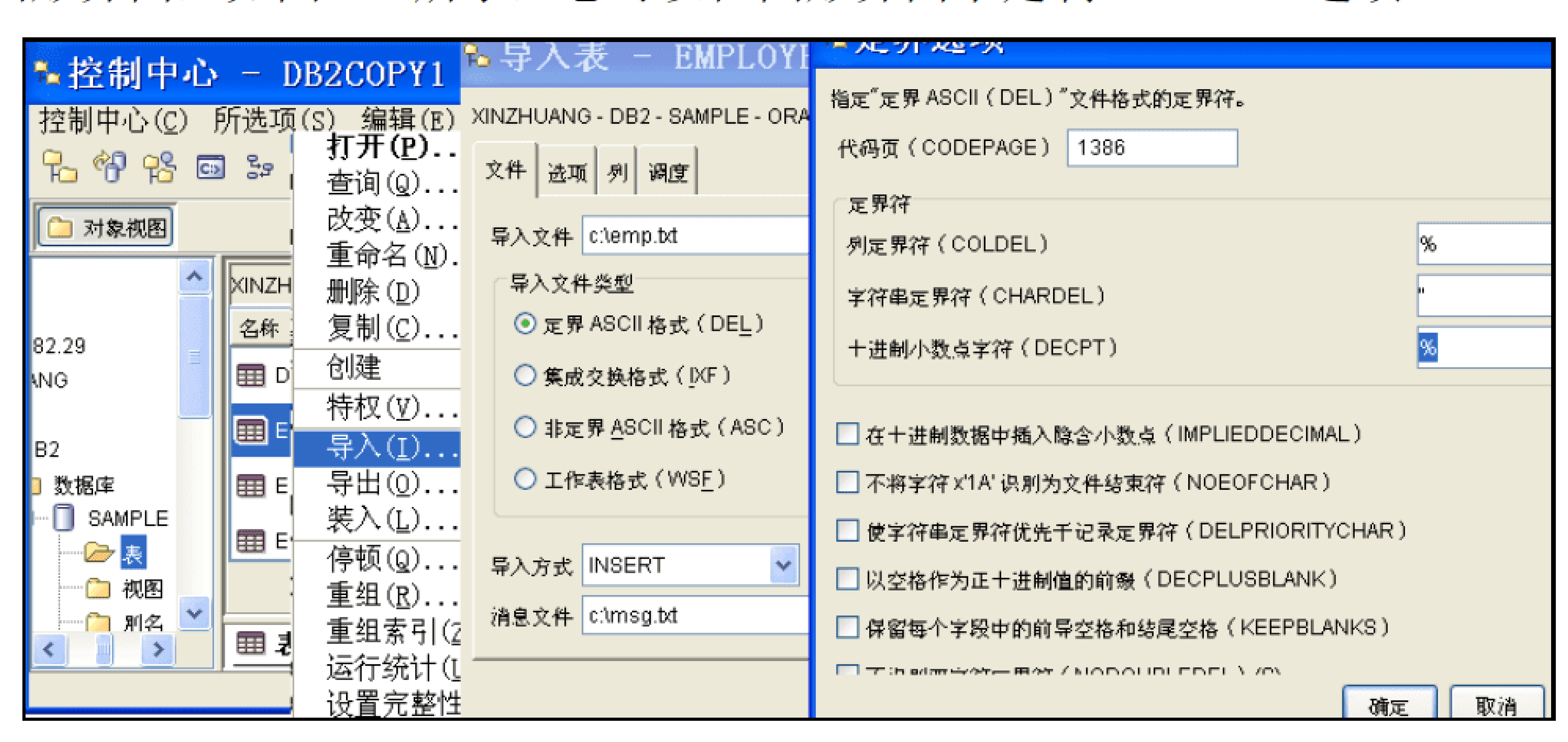


图 6-4 使用控制中心导入数据

2. IMPORT 命令

IMPORT 常用命令参数如图 6-5 所示。

● filename

指定进行数据导入的文件。

● OF filetype

指定导入文件的类型。导入上面提到的 4 种支持文件格式:DEL、WSF、IXF 和 ASC。

● LOBS FROM lob-path

如果表中有 LOB 类型数据需要导入，就指定一个或多个目录文件，用于存储 LOB 数据。

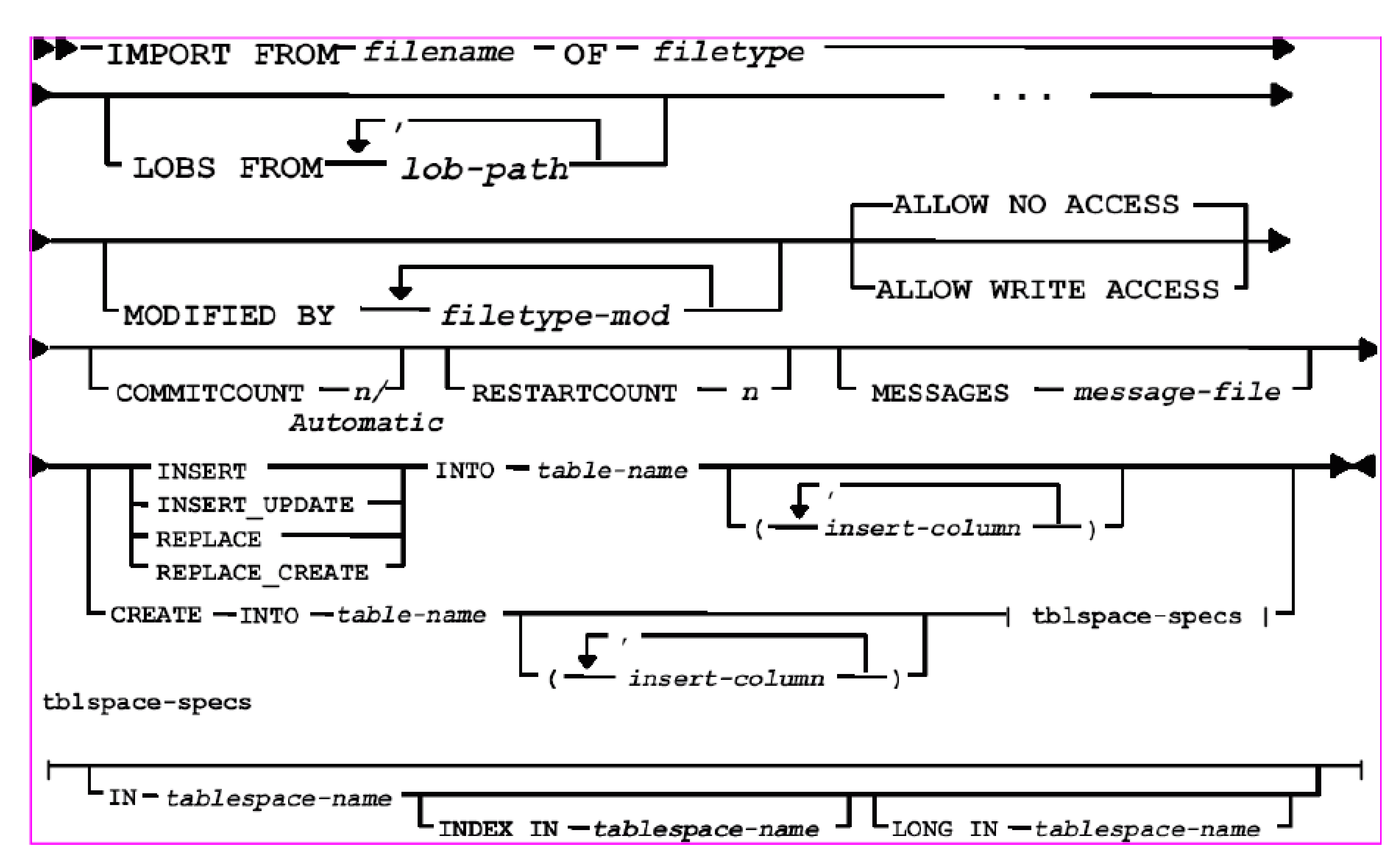


图 6-5　IMPORT 常用命令参数

- MODIFIED BY filetype-mod

指定一些额外的文件类型修饰符参数。文件类型修饰符提供了允许您更改数据、日期和时间戳记或代码页格式之类的许多选项。

- 脱机导入(ALLOW NO ACCESS)

在 ALLOW NO ACCESS 方式下，导入会在插入任何行之前获取针对目标表的独占(X)锁定。挂起对该表的锁定有两种影响：

◊ 首先，如果其他应用程序挂起针对导入目标表的表锁定或行锁定，那么 IMPORT 实用程序将等待这些应用程序落实或回滚更改。

◊ 其次，IMPORT 实用程序运行时，请求锁定的任何其他应用程序会等待导入操作完成。

注意：

与锁相关的知识点，可以参见本书第 10 章内容。读者也可以读完第 10 章后再回过头来看这一部分和锁相关的 IMPORT 内容。

- 联机导入(ALLOW WRITE ACCESS)

在 ALLOW WRITE ACCESS 方式下，IMPORT 实用程序将获取针对目标表的非独占(IX)锁定。挂起对该表的此锁定具有下列影响：

◊ 如果其他应用程序挂起不兼容的表锁定，那么在所有这些应用程序落实或回滚更改之前，IMPORT 实用程序不会开始插入数据。

◊ IMPORT 实用程序运行时，如果任何其他应用程序请求不可兼容的表锁定，那么这些应用程序都将等待直至导入操作落实或回滚当前事务。注意，导入的表锁定仅对单个事务有效。因此，在每次提交后，联机导入必须请求表锁定并可能需要等待。
◊ 如果其他应用程序挂起不兼容的行锁定，那么 IMPORT 实用程序将停止插入数据直到所有这些应用程序落实或回滚更改。
◊ IMPORT 实用程序运行时，如果任何其他应用程序请求不可兼容的行锁定，那么这些应用程序都将等待直至导入操作落实或回滚当前事务。
◊ 为保留联机属性并降低死锁几率，ALLOW WRITE ACCESS 导入将定期落实当前事务，并在上升为独占表锁定之前释放所有行锁定。如果未显式设置提交频率，那么导入会按指定 COMMITCOUNT AUTOMATIC 的方式落实。如果 COMMITCOUNT 设置为 0，那么不会执行任何落实。

● COMMITCOUNT n

每隔 n 条记录进行一次提交。避免在出现错误以后，需要重新导入所有的数据。比如，设定 n 为 100，那么系统每隔 100 条记录就进行一次提交，将导入的数据保存下来。如果在导入到 531 条记录时出现了错误，因为已经保留了前面的 500 条记录，只需要从第 501 条记录开始导入即可。也可以设置为 AUTOMATIC，让 DB2 自动设置 COMMITCOUNT。

● RESTARTCOUNT n

一般用于在导入过程失败了之后，定义重新进行导入的起点。对于前面的例子而言，n 应该设定为 501。

● MESSAGES message-file

用于指定在导入表过程中生成的警告信息和错误信息的存储文件，如果没有指定，将导出信息到标准导出上，如屏幕。一般来说，在导入完成后，应该检查这个文件中的信息。其中，比较重要的信息包括要导入的数据行数目、成功导入的行的数目，以及被拒绝导入的行的数目。

● 导入方式

导入可使用 5 种方式，它们用于确定导入数据的方法。前 3 种方式为 INSERT、INSERT_UPDATE 和 REPLACE，在目标表已存在的情况下使用，如表 6-2 所示。这 3 种方式都支持 IXF、WSF、ASC 和 DEL 数据格式。但是，只有 INSERT 和 INSERT_UPDATE 可与昵称(nickname)配合使用。

表 6-2　导入数据的前 3 种方式

方　　式	工 作 机 制
INSERT	将输入数据插入到目标表中而不更改现有数据
INSERT_UPDATE	将导入的数据行插入到表中，如果导入的数据与表中原来的数据主键一样，就执行更新(update)操作，否则执行插入(insert)操作。表中有主键时，才可以使用这种模式
REPLACE	删除所有现有数据并插入已导入数据，同时保留表和索引定义

另外两种方式为 REPLACE_CREATE 和 CREATE，在目标表不存在时使用，如表 6-3 所示。它们只能与 PC/IXF 格式的输入文件配合使用，此格式包含要创建的表的结构化描述。如果对象表具有自身以外的任何从属，那么不能以这些方式执行导入。

注意：
不建议使用导入的 CREATE 和 REPLACE_CREATE 方式。请改用 db2look 实用程序。

表 6-3　导入数据的后两种方式

方　　式	最佳实践用法
REPLACE_CREATE	删除所有现有数据并插入已导入数据，同时保留表和索引定义，如果目标表和索引不存在，那么创建目标表和索引
CREATE	创建目标表和索引，可指定在其中创建新表的表空间名称

下面是一个使用 CLP 导入数据的例子：

```
db2 import from employee.txt of del messages msg.out  insert into employee
```

3. 使用 ADMIN_CMD 存储过程

可以在命令行中直接调用 ADMIN_CMD 存储过程导出数据，如下所示：

```
  call sysproc.admin cmd('import from /home/db2inst1/output/sales.del of del
messages /home/db2inst1/output/export.msg insert into sales')
```

导入的工作机制

导入所需的步骤数和时间量取决于要移动的数据量和指定的选项。导入操作遵循下列步骤：

(1) 锁定表

根据您是否允许对表进行并行访问，导入会获取对现有目标表的独占(X)或非独占(IX)锁定(关于锁这部分信息，大家先简单了解，本书第 10 章会详细讲解锁)。

IMPORT 实用程序支持两种表锁定方式：脱机或 ALLOW NO ACCESS 方式；以及联机或 ALLOW WRITE ACCESS 方式。ALLOW NO ACCESS 方式会阻止并行应用程序访问表数据。ALLOW WRITE ACCESS 方式允许并行应用程序同时对导入目标表进行读写访问。如果未显式指定任何方式，那么导入会以默认方式 ALLOW NO ACCESS 运行。同时，在默认情况下，IMPORT 实用程序会使用隔离级别 RS(读稳定性)绑定至数据库。

(2) 查找和检索数据

导入使用 FROM 子句来查找输入数据。如果命令指示 XML 或 LOB 数据存在，那么导入会查找此数据。

(3) 插入数据

导入会替换现有数据或将新的数据行添加至表。

(4) 检查约束和激发触发器

写入数据后，导入会确保每个已插入行符合针对目标表定义的约束。有关被拒绝行的信息将写至消息文件。导入还会激发现有触发器。

(5) 提交操作

导入会保存所作更改并释放针对目标表的锁定。还可指定在导入期间定期落实。

6.3.3　导入数据示例

例 6-7　使用命令 CLP 导入 XML 文档。

```
IMPORT FROM "C:\XML\data\import.del"
OF DEL XML FROM "D:\XML\data" METHOD P (1)
   MESSAGES "C:\XML\xmlemp1.log"      INSERT INTO DB2ADMIN.XMLEMP (EMP);
```

其中的 D:\XML\data\import.del 包含指向实际文档的行指针。import.del 示例文件的内容如下：

```
 "<XDS FIL='emp.001.xml' />"
     "<XDS FIL='emp.002.xml' />"
     "<XDS FIL='emp.003.xml' />"
     "<XDS FIL='emp.004.xml' />"
     "<XDS FIL='emp.005.xml' />"
     "<XDS FIL='emp.006.xml' />"
     "<XDS FIL='emp.007.xml' />"
```

在成功装载之后，应该会在消息文件中看到下面这样的消息：

```
SQL3109N  The utility is beginning to load data from file
"D:\XMLPoT\labdoc\scripts\data\import.del".
SQL3110N  The utility has completed processing.
"42" rows were read from the input file.
SQL3221W  ...Begin COMMIT WORK. Input Record Count = "42".
SQL3222W  ...COMMIT of any database changes was successful.
SQL3149N  "42" rows were processed from the input file.
"42" rows were successfully inserted into the table.
"0" rows were rejected.
```

例 6-8　以下示例显示导入以管道符分界的文件：

```
IMPORT FROM "D:\db2out\employee.txt" OF DEL MODIFIED BY COLDEL|
MESSAGES "D:\db2out\employee2.log"      INSERT INTO DB2ADMIN.EMPLOYEE
```

IMPORT 实用程序存在下列限制：

- 如果现有表是一个父表，并且它包含的主键被从属表中的外键引用，那么不能替换此表的数据，而只能追加数据。
- 不能执行导入替换操作来将数据导入到以立即刷新方式定义的物化查询表所关联的基表中。
- 不能将数据导入到系统表、摘要表或其他带有结构化类型列的表中。
- 不能将数据导入到已声明临时表中。
- 不能通过 IMPORT 实用程序创建视图。
- 根据 PC/IXF 文件创建表时，不会保留引用约束和外键定义。如果数据先前是使用 SELECT * 导出的，那么会保留主键定义。
- 由于 IMPORT 实用程序会生成自己的 SQL 语句，所以在某些情况下可能会超过最大语句大小(即 2MB)。
- 不能使用 CREATE 或 REPLACE_CREATE 导入选项重新创建分区表或多维集群表(MDC)。
- 不能重新创建包含 XML 列的表。
- 不能导入已加密的数据。
- 导入替换操作不能识别 NLI(Not Logged Initially)子句。IMPORT 命令的 REPLACE 选项不能识别 CREATE TABLE 语句的 NOT LOGGED INITIALLY (NLI)子句或 ALTER TABLE 语句的 ACTIVATE NOT LOGGED INITIALLY 子句。如果包含 REPLACE 操作的导入操作与调用了 NLI 子句的 CREATE TABLE 或 ALTER TABLE

语句在同一事务内执行，那么此导入操作不能识别该 NLI 子句。将记录所有插入操作。可以使用以下两种变通方法：

◊ 使用 DELETE 语句删除表的内容，然后使用 INSERT 语句调用导入操作。

◊ DROP 后重新创建该表，接着使用 INSERT 语句调用导入操作。

6.4 LOAD

6.4.1 LOAD 概述

我们在前面已经讲解了 IMPORT 实用程序，IMPORT 本质上是执行 INSERT、UPDATE 和 DELETE SQL 语句。所以在把数据放到表中时，会触发触发器、执行日志记录并执行约束检查和索引构建。由于 LOAD 实用程序直接将格式化的数据页(datapage)写入数据库，这会绕过触发器和日志记录机制，所以 LOAD 实用程序不会触发触发器，并且除了验证索引唯一性以外不执行引用约束检查或表约束检查。LOAD 实用程序能够高效地将大量数据移到新创建的表或者已包含数据的表中。此实用程序能够处理绝大多数数据类型，其中包括 XML、大对象(LOB)和用户定义的类型(UDT)。下面我们来介绍 LOAD。

LOAD 过程由四个不同的阶段组成如图 6-6 所示。

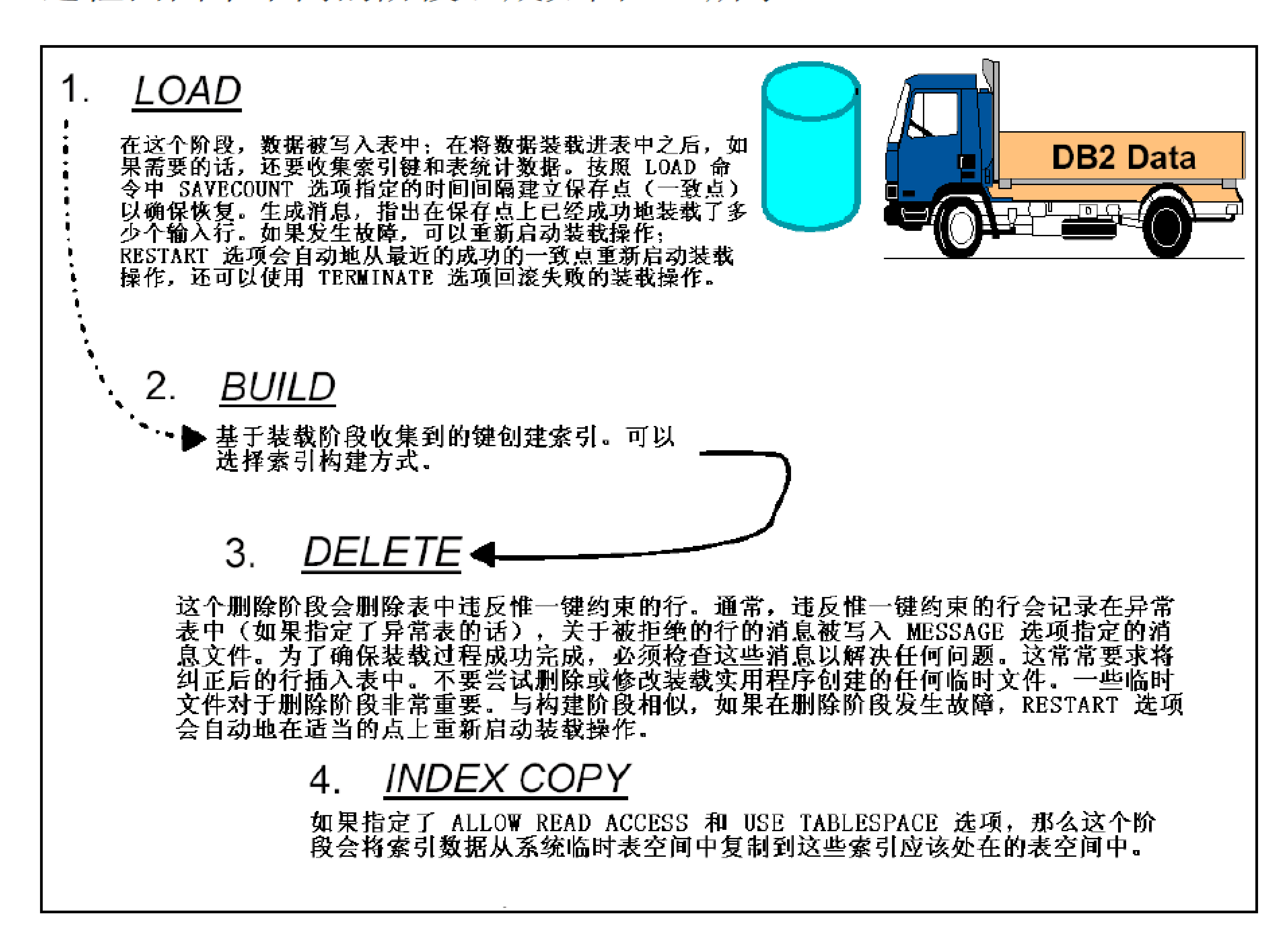


图 6-6　LOAD 过程

6.4.2 装入数据

LOAD 实用程序能够高效地将大量数据移到新创建的表或者已包含数据的表中。在调用 LOAD 实用程序前，您必须连接至(或者能够隐式地连接至)要装入数据的数据库。由于该实用程序将发出 COMMIT 语句，所以应该通过发出 COMMIT 或 ROLLBACK 语句来完成所有事务并释放所有锁定，然后再调用 load 实用程序。

与 DB2 Import 相似，DB2 LOAD 要求授予某些权限和特权。需要 SYSADM 或 DBADM 权限，至少要有 LOAD 权限和相关的 INSERT 或 DELETE 特权。

可以通过命令行处理器(CLP)、控制中心中的“装入”向导或者调用 ADMIN_CMD 存储过程来调用 LOAD 实用程序。

1. 使用控制中心 LOAD 数据

尽管命令行方式灵活而且强大，但是在命令提示语法中选择众多的选项是很麻烦的。调用 LOAD 的一个简便方法是使用“控制中心”，它会以向导驱动的方式为用户提供在线帮助。“控制中心”可以引导用户轻松地实现成功的装载，即使用户不熟悉 LOAD 也没关系。在下面的步骤中，将一个分界的文件装载进表 *employee* 中，从而体会一下如何在“控制中心”中进行数据装载。

(1) 选择表 *employee* 并选择“装入”，从而调用 LOAD。这时会显示图 6-7 所示的对话框。

注意，这个向导有 8 个步骤。但是，可以直接单击“下一步”选用默认设置。每个步骤都提供了不同的选项，根据您的装载需求，其中一些选项是必需的。在这个简单的演示中，第一步采用默认的“将数据附加到表中”。在这种模式下，允许用户在装载期间访问数据。单击“下一步”继续。

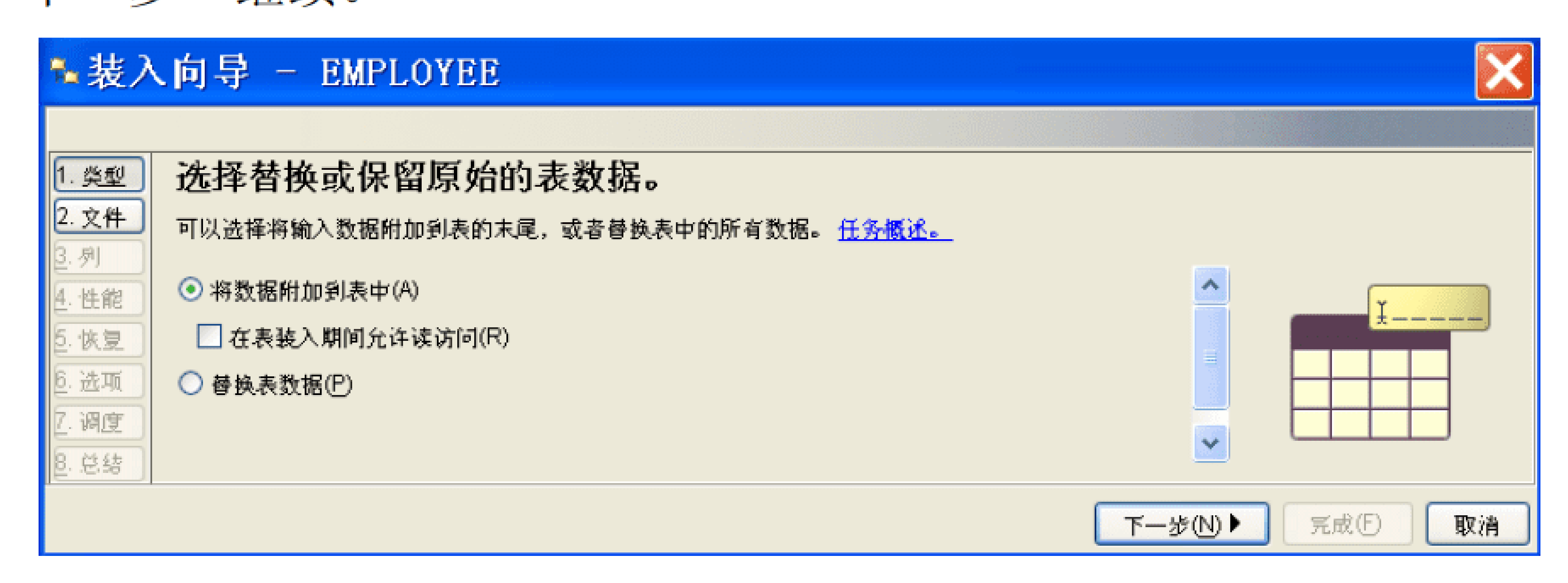


图 6-7 选择追加还是替换

(2) 第二步带领用户选择文件格式(默认格式是 DEL)。在这一步中，选择输入文件和消息文件的位置——本地(即调用 LOAD 的地方)还是远程。还可以指定要处理的总行数。

单击“下一步”继续。您还可以定制 DEL 选项，如图 6-8 所示。

装入向导 － EMPLOYEE
1. 类型
2. 文件
3. 列
4. 性能
5. 恢复
6. 选项
7. 调度
8. 总结
指定输入和输出文件。
大部分装入操作将至少有一个输入或输出文件。可以在
定界文本（DEL）
DEL 选项
输入文件位置
服务器(S) (XINZHUAN)
远程主机(R)
数据库服务器上的用户出口程序的完整路径和文件名：
输入文件的完整路径和文件名
c:\emp.txt
要存储进度消息的完整路径和文件名：
c:\emp.msg

定界选项
指定“定界 ASCII（DEL）”文件格式的定界符。
代码页（CODEPAGE） 1386
定界符
列定界符（COLDEL） <缺省值>
字符串定界符（CHARDEL） <缺省值>
十进制小数点字符（DECPT） <缺省值>
在十进制数据中插入隐含小数点（IMPLIEDDECIMAL）
不将字符 x'1A' 识别为文件结束符（NOEOFCHAR）
使字符串定界符优先于记录定界符（DELPRIORITYCHAR）
以空格作为正十进制值的前缀（DECPLUSBLANK）
保留每个字段中的前导空格和结尾空格（KEEPBLANKS）
不识别双字符定界符（NODOUBLEDEL）(S)
确定　取消　帮助

图 6-8　指定文件位置

(3) 在这一步中，可以指定 LOB 对象的位置。可以指定标识列和生成列行为。另外，可以选择在装载中包含哪些列，如图 6-9 所示。接受默认设置并单击“下一步”。



图 6-9　指定 LOB 对象的位置

(4) 对于索引创建有 3 个选项：递增式、重新构建，或者让 LOAD 自己决定构建索引的最佳方式，如图 6-10 所示。有几个应用程序控制级别，而且 LOAD 有能力使装载后的性能达到最优。

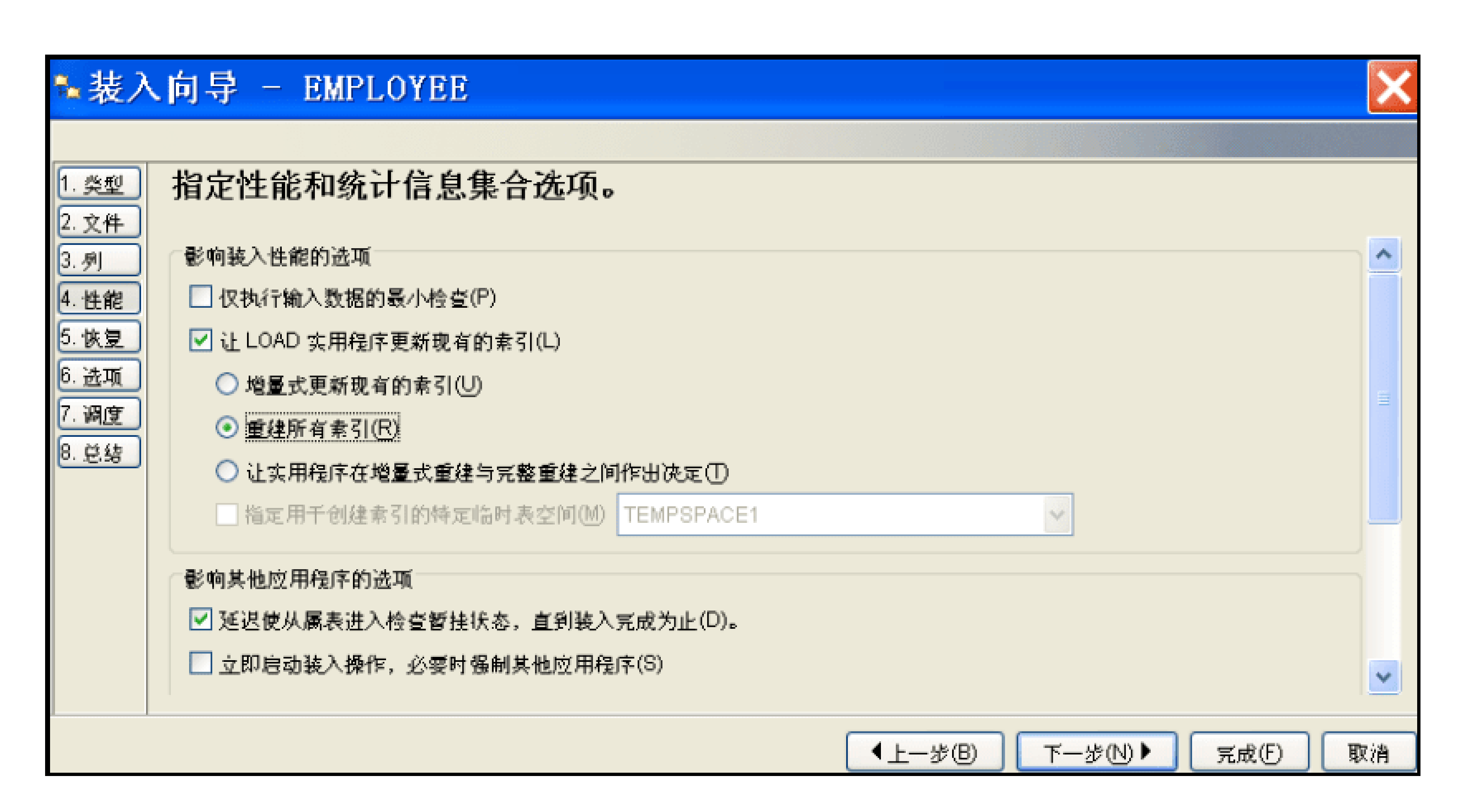


图 6-10　索引、应用程序和性能选项

(5) 在 LOAD 运行期间，系统可能会崩溃。为了能够从系统崩溃中恢复，LOAD 提供了崩溃恢复选项，使用户能够指定一致点(savecount)，如图 6-11 所示。因为 LOAD 在事务期间很少进行日志记录，所以有可能需要进行向前恢复。在这一步中，可以选择可恢复(在这里可以保存备份映像的副本)，或者选择不可恢复从而不允许在发生故障时进行恢复。

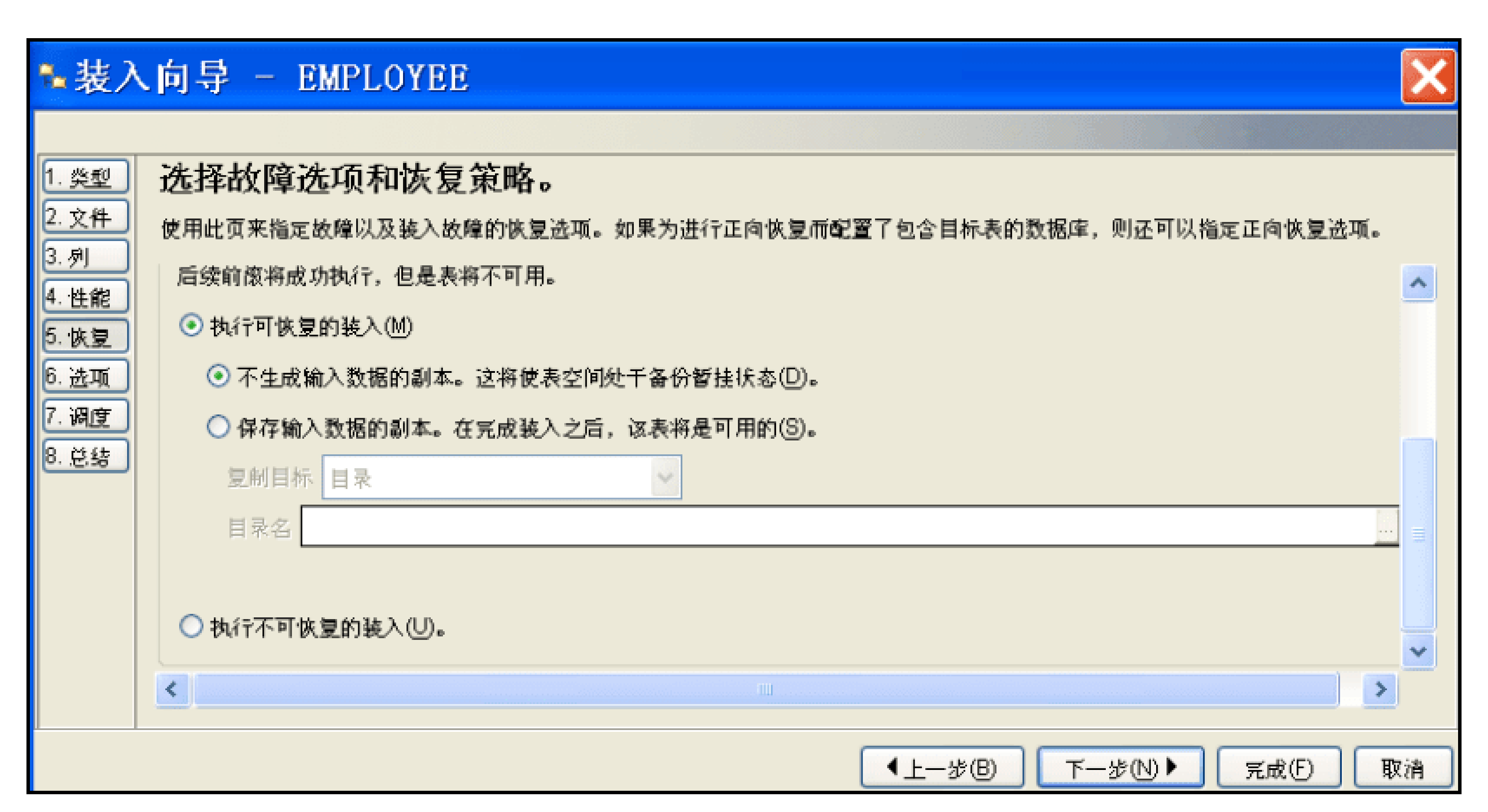


图 6-11　恢复选项

(6) 如果您不能确定合适的性能参数，那么明智的做法是让 DB2 配置顾问进行选择。在这里可以设置的其他选项是异常表的位置，包含被拒绝的行的异常转储文件，以及抑制接收任何警告，如图 6-12 所示。

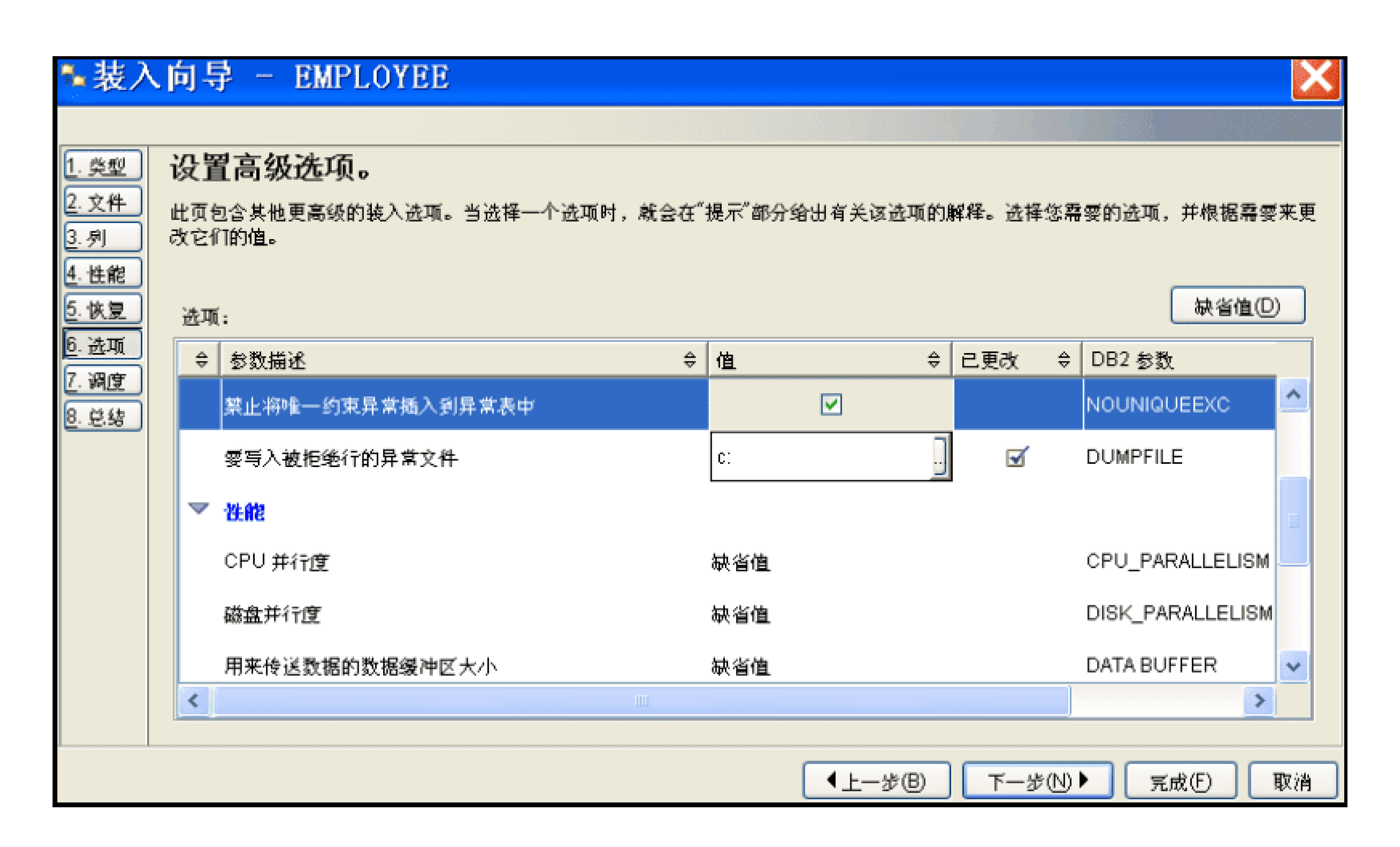


图 6-12　更多选项

步骤(7)和(8)允许进行调度(就像对 IMPORT 进行调度一样)。最后一步是总结您已经选择的选项。最后(但是并非不重要)，您应该注意 DB2 LOAD 有以下限制：

- 不支持将数据装载进昵称(nickname)、层次结构、有类型表、声明的临时表、包含 XML 列的表或具有结构化类型列的表。
- 如果在 LOAD REPLACE 操作期间发生错误，那么表中原来的数据会丢失。保护措施是保留输入数据的副本，从而允许在发生故障时重新启动装载操作。
- 在新装载的行上不激活触发器。装载实用程序并不实施与触发器相关联的业务规则。

2. 使用 LOAD 命令

```
LOAD [CLIENT] FROM file/pipe/dev/cursor name [ {,file/pipe/dev} ... ]
    OF {ASC | DEL | IXF | CURSOR}
    [LOBS FROM lob-path [ {,lob-path} ... ] ]
    [MODIFIED BY filetype-mod [ {filetype-mod} ... ] ]]
    [METHOD {L ( col-start col-end [ {,col-start col-end} ... ] )
              [NULL INDICATORS (col-position [ {,col-position} ... ] )]
          | N ( col-name [ {,col-name} ... ] )
          | P ( col-position  [ {,col-position} ... ] )}]
    [SAVECOUNT n]
    [ROWCOUNT n] [WARNINGCOUNT n] [MESSAGES msg-file]
    [TEMPFILES PATH pathname]
    {INSERT | REPLACE | RESTART | TERMINATE}
    INTO table-name [( insert-column [ {,insert-column} ... ] )]
    [FOR EXCEPTION table-name [NOUNIQUEEXC NORANGEEXC]]
```

```
        [STATISTICS {NO | USE PROFILE}]
        [{COPY {NO  | YES { USE TSM [OPEN num-sess SESSIONS]
                    | TO dir/dev [ {,dir/dev} ... ]
                    | LOAD lib-name [OPEN num-sess SESSIONS]}}
                    | NONRECOVERABLE} ]
        [HOLD QUIESCE] [WITHOUT PROMPTING] [DATA BUFFER buffer-size]
        [SORT BUFFER buffer-size] [CPU_PARALLELISM n] [DISK_PARALLELISM n]
        [INDEXING MODE {AUTOSELECT | REBUILD | INCREMENTAL | DEFERRED}]
        [SET INTEGRITY PENDING CASCADE {DEFERRED | IMMEDIATE}]
        [ALLOW NO ACCESS | ALLOW READ ACCESS [USE tblspace-name]] [LOCK WITH
FORCE]
        [[PARTITIONED DB CONFIG] partitioned-db-option
[{partitioned-db-option}...]]
         filetype-mod:
          NOROWWARNINGS, ANYORDER, BINARYNUMERICS, CODEPAGE=x,
          DUMPFILE=x, FASTPARSE, NOHEADER, TOTALFREESPACE=x,
          INDEXFREESPACE=x, PAGEFREESPACE=x, FORCEIN, IMPLIEDDECIMAL,
          PACKEDDECIMAL, NOCHECKLENGTHS, NOEOFCHAR, NULLINDCHAR=x,
          RECLEN=x, STRIPTBLANKS, STRIPTNULLS, NODOUBLEDEL, LOBSINFILE,
          CHARDELx, COLDELx, DLDELx, DECPLUSBLANK, DECPTx, DATESISO,
          DELPRIORITYCHAR, USEDEFAULTS, DATEFORMAT=x, TIMEFORMAT=x,
          TIMESTAMPFORMAT=x, ZONEDDECIMAL, KEEPBLANKS, IDENTITYMISSING,
          IDENTITYIGNORE, IDENTITYOVERRIDE, GENERATEDMISSING,
          GENERATEDIGNORE, GENERATEDOVERRIDE, USEGRAPHICCODEPAGE
         partitioned-db-option:
          HOSTNAME x, FILE TRANSFER CMD x, PART FILE LOCATION x,
OUTPUT_DBPARTNUMS x,
          PARTITIONING DBPARTNUMS x, MODE x, MAX NUM PART AGENTS x,
OMIT_HEADER,
          ISOLATE PART ERRS x, STATUS INTERVAL x, PORT RANGE x,
CHECK_TRUNCATION,
          MAP_FILE_INPUT x, MAP_FILE_OUTPUT x, TRACE x, NEWLINE, DISTFILE x
```

下面我们来看一个 LOAD 的例子：

DB2 LOAD 命令文件示例(INSERT)	DB2 LOAD 命令文件示例(REPLACE)
(1) load	(1) LOAD
(2) FROM 'INPUT_FILE1.DAT'	(2) FROM 'INPUT_FILE2.DAT'
(3) OF ASC	(3) OF DEL
(4) MODIFIED BY DUMPFILE='INPUT_FILE1.BAD'	(4) MODIFIED BY DUMPFILE='INPUT_FILE2.BAD'
(5) METHOD L (1 5, 6 15, 16 20)	(3) COLDEL|

（续表）

DB2 LOAD 命令文件示例(INSERT)	DB2 LOAD 命令文件示例(REPLACE)
(6) INSERT INTO PROD.TB_TABLE1 (7) (COL1, COL2, COL3) (8) FOR EXCEPTION PROD.TB_TABLE1_DSC;	(3) CHARDEL" (5) METHOD P (1, 2, 3) (6) REPLACE INTO PROD.TB_TABLE2 (7) (COL1, COL2, COL3) (8) FOR EXCEPTION PROD.TB_TABLE2_DSC;

下面我们解释上面例子锁用到的命令选项：

(1) LOAD

这会在 DB2 中调用 LOAD 实用程序。

(2) FROM [inputfile_name]

这是包含要装载的数据的文件。DB2 LOAD 还可以从管道、设备或游标装载数据。

(3) OF ASC/ DEL

对于 DB2 LOAD，ASC 表示不分界的 ASCII 数据，数据的划分由位置决定。DEL 表示分界的 ASCII 数据，每行的数据长度可变。分界的数据可以使用多种修饰符，主要的两种是 COLDEL 和 CHARDEL；COLDEL 决定列和列之间如何分界，默认设置是逗号；CHARDEL 决定字符串数据如何分界，默认设置是双引号。

(4) MODIFIED BY DUMPFILE=[dumpfile_name]

DB2 把被拒绝的记录放到这个文件中。

(5) METHOD P (1，2，3)

DB2 LOAD 有 3 个方法：

- METHOD L 只用于 ASC 数据，这个方法要指出每列的开头和结尾。它的形式是：METHOD L (start1 end1，start2 end2…)。
- METHOD N 用于 IXF 或游标数据，它要指定源表中要装载的列。它的形式是：METHOD N (col1，col2，col4…)。
- METHOD P 用于 DEL、IXF 或游标数据，它要指定源数据中要装载的列的位置(position)号。它的形式是：METHOD P (1，2，4…)。

(6) INSERT / REPLACE INTO PROD.TABLE

DB2 LOAD 在这里有两个选项:INSERT 和 REPLACE。另外两个 DB2 LOAD 选项是 RESTART 和 TERMINATE。当 DB2 LOAD 由于任何原因未完成时，使用这些选项。

(7) (COL1，COL2，COL3) Insert Column List

DB2 LOAD 使用这个列表决定要放入数据的列。如果省略这个列表，那么 DB2 LOAD

会尝试按照读取和解析数据的次序装载数据。

(8) FOR EXCEPTION [table_name]

DB2 LOAD 把违反唯一约束和主键规则的记录(异常)放到以前创建的这个表中。

LOAD 命令非常复杂，基本上是 DB2 中最复杂的命令，有很多命令选项，表 6-4 总结了常用的命令选项。还有一些选项我们会在后面特定的 LOAD 性能中讲到。

表 6-4　常用的 LOAD 命令选项

LOAD 命令关键字	LOAD 命令关键字解释
WARNINGCOUNT=number	使用此参数来指定强制装入操作终止之前该实用程序可返回的警告数目。如果您只需要很少警告或不需要警告，那么将 WARNINGCOUNT 参数设置为相对较小的数字。装入操作将在达到 WARNINGCOUNT 数目时停止。这允许您在尝试完成装入操作之前解决问题
NOROWWARNINGS	在装入操作期间，关于已拒绝的行的警告消息将写入指定的文件中。但是，如果 LOAD 实用程序必须处理大量已拒绝的、无效或已截断的记录，那么可能会对装入性能产生负面影响。如果预计到会产生许多警告，那么使用 norowwarnings 文件类型修饰符来抑制记录这些警告会很有用。但是这样做可能有少许风险
MESSAGES messagefile	DB2 把消息放到这个消息文件中。如果不指定消息文件，它就不产生消息。为了定位导入期间产生的问题，建议指定消息文件
ROWCOUNT number	指定要装载的记录数。如果省略这个关键字，那么默认设置是所有记录
CPU_PARALLELISM number DISK_PARALLELISM number FETCH_PARALLELISM yes	DB2 V9 中 LOAD 自动决定这些设置，用来控制对文件、设备、管道和游标装载中的记录进行解析、转换、格式化和写操作所生成的线程数。也可以使用这些关键字指定自己需要的值。可以提高性能
DATA BUFFER number	LOAD 使用许多大小为 4KB 的页面传输数据，这个数值通常是自动决定的，但是也可以用这个关键字指定自己需要的值
SAVECOUNT number	LOAD 使用一致点确保装载操作的可恢复性
RESTART REPLACE, INSERT, TERMINATE	LOAD 使用 restart 模式在遇到故障之前的最后一个一致点之后选择重新装载的起始点。LOAD 会自己决定起始点，不需要操作者计算。REPLACE 覆盖已有数据、INSER 追加数据，TERMINATE 终止 LOAD 程序

(续表)

LOAD 命令关键字	LOAD 命令关键字解释
INDEXING MODE DEFERRED	该模式意味着在装载期间不会创建索引。涉及的索引上会作出标记，但是需要刷新。当重新启动数据库或者第一次访问那些索引时，才会重新构建那些索引
INDEXING MODE REBUILD，INCREMENTAL，AUTOSELECT	REBUILD 模式强制重新构建所有索引。INCREMENTAL 模式只向索引中添加新增的数据。AUTOSELECT 模式允许实用程序在 REBUILD 和 INCREMENTAL 之间作出选择
NONRECOVERABLE 和 COPY NO 区别	如果启用了前滚恢复，那么在前滚后不需要对表恢复装入事务的情况下使用此参数。NONRECOVERABLE 装入和 COPY NO 装入具有完全相同的性能。但是，在潜在数据丢失方面却有重大差别。NONRECOVERABLE 装入将表标记为不可前滚恢复，并同时使得能够完全访问表。这可能会产生一个问题，在需要前滚装入操作的情况下，已装入的数据以及所有对表的后续更新都会丢失。COPY NO 装入使所有从属表空间处于“备份暂挂”状态，这将导致在执行备份之前，表不可访问。因为在该类型的装入后会强制执行表空间备份，所以不存在丢失已装入数据或对表的后续更新的风险。也就是说，COPY NO 装入完全可恢复。如果使用这个选项，在装载操作之后表空间处于 backup pending 状态，在装载操作期间不复制(copy)装载的数据
COPY YES	使用此参数来指定在装入操作期间是否创建输入数据的副本。仅当启用了归档日志时，COPY YES 才适用。COPY NO 选项会导致与装入的表相关的表空间将处于“备份暂挂”状态，并且必须先备份这些表空间才能访问该表。而 COPY Yes 在 LOAD 期间自动做表空间备份
MODIFIED BY RECLEN=x	固定的输入记录的大小。适用于 ASC 文件格式
MODIFIED BY DUMPFILE=filename	指定一个文件用来决定在哪里存储被拒绝(reject)的记录
FOR EXCEPTION tablename	异常表存储不遵循唯一约束和主键约束的行。如果装入的时候没有指定异常表，则违反唯一约束的行将被丢弃并且不再有机会恢复或修改。异常表结构比要装入表的结构增加了两列：时间戳列和消息描述列 CLOB(32KB)
OF DEL，ASC，IXF 和 WSF	LOAD 数据格式

(续表)

LOAD 命令关键字	LOAD 命令关键字解释
INTO TABLE tablename	要存储数据的目标表
MODIFIED BY COLDELx	在所有输入数据中以这个字符分隔各个列(默认设置是逗号)
MODIFIED BY CHARDELx	用这个字符包围输入的字符数据(默认设置是双引号)
1. LOBS FROM pathnames 2. MODIFIED BY LOBSINFILE	寻找 LOB 文件的路径。要想使用 LOBS FROM 子句，就必须设置这个关键字。参数本身包含文件名，但是不包含完整的路径名，因为将搜索 LOBS FROM 路径
ALLOW NO ACCESS/ ALLOW READ ACCESS	在装入操作执行过程中，是否可以查询表中预先存在的数据。ACCESS READ ACCESS 选项允许您在进行装入操作时查询表。只能查看装入操作之前表中已存在的数据
LOCK WITH FORCE	装入操作在继续执行之前是应该等待其他实用程序或应用程序使用完表，还是应该强制其他应用程序释放锁定
STATISTICS USE PROFILE	在装入过程中是否收集统计信息。仅当以 REPLACE 方式运行装入操作时，才支持此选项
TEMPFILES PATH	装入操作执行期间创建临时文件时要使用的标准路径。此名称由 LOAD 命令的 TEMPFILES PATH 参数指定。默认值是数据库路径。并且由 DB2 实例以独占方式访问
USE <tablespace-name>	如果正在执行 ALLOW READ ACCESS 装入并且建立索引方式为 REBUILD，那么此参数允许在系统临时表空间中重建索引，并在装入操作的索引复制阶段将其复制回索引表空间

3. 使用 ADMIN_CMD 存储过程

可以在命令行中直接调用 ADMIN_CMD 存储过程导出数据，如下所示：

```
    call sysproc.admin_cmd('load from /home/db2inst1/output/sales.del of del
messages /home/db2inst1/output/load.msg replace into sales')
```

6.4.3 装入示例

例 6-9 看看下面这个例子，它演示了一个装载过程中涉及的步骤：

```
    LOAD FROM emp.ixf OF IXF   MESSAGES msg.out   MODIFIED BY
DUMPFILE=c:\emp.dmp
```

```
    TEMPFILES PATH d:\tmp    INSERT INTO employee    FOR EXCEPTION empexp
```

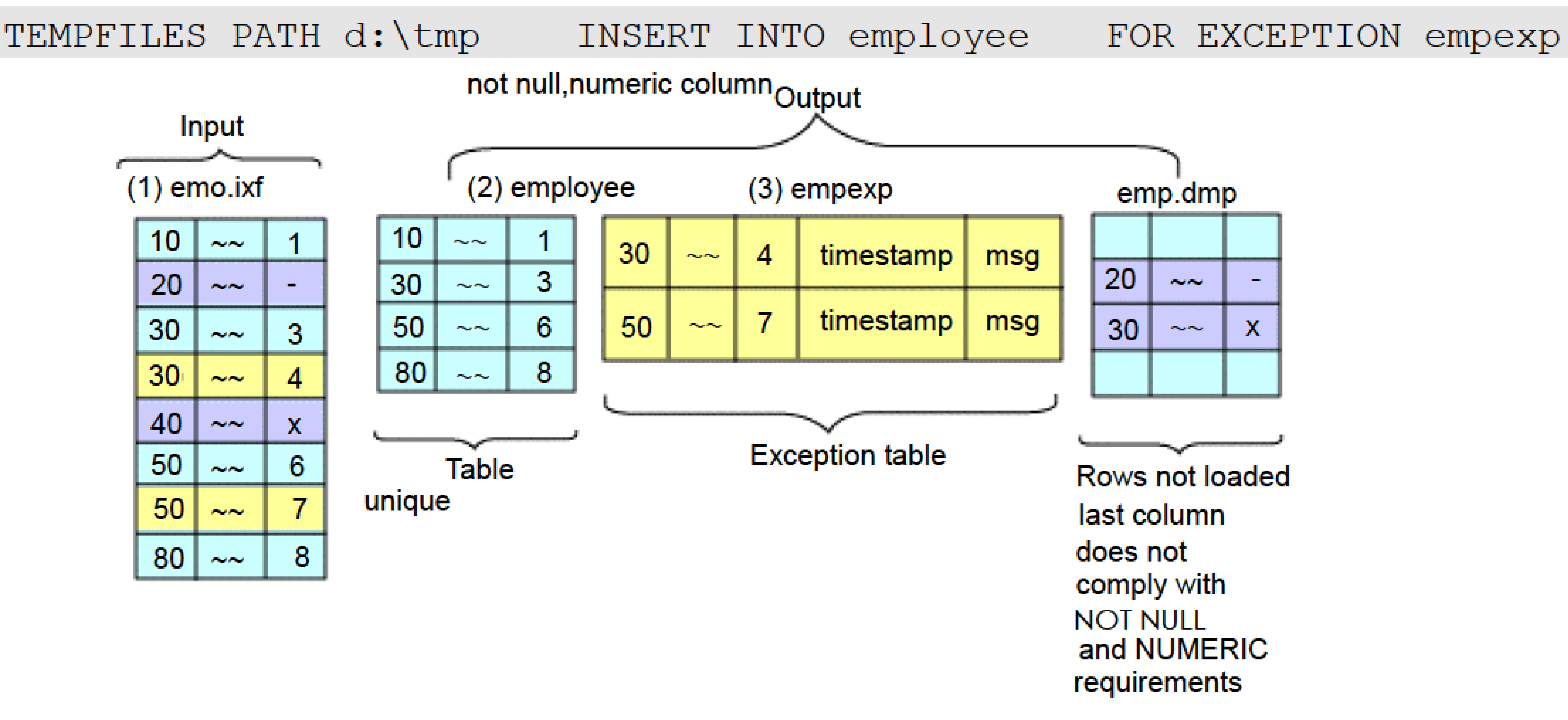


图 6-13　装载过程

在图 6-13 中

(1) 显示了输入源文件的内容。

(2) 中显示的目标表 EMPLOYEE 是用以下列定义创建的：

第一列必须是唯一的。最后一列是一个数值列，且不能为 NULL。

(3) 中显示的异常表 EMPEXP 是使用和 EMPLOYEE 相同的列，再加上时间戳和消息列创建的。

在装载阶段，输入文件中的所有数据被装载到 EMPLOYEE 中，除了标为粉色的两个行(20 和 40)，因为它们不符合 NOT NULL 和 NUMERIC 列定义。由于指定了 DUMPFILE 修饰符，因此这两行的数据被记录在 C:\emp.dmp 文件中。

在删除阶段，标为黄色的两个行(30 和 50)被从 EMPLOYEE 中删除，并插入到异常表 EMPEXP 中。这是因为它们违反了 EMPLOYEE 表中第一列的唯一性约束。

在装载的最后，您应该检查消息文件、转储文件和异常表，然后决定如何处理被拒绝的行。如果装载成功完成，那么在 D:\tmp 中生成的临时文件将自动被删除。

例 6-10　(使用转储文件)

表 FRIENDS 的定义如下所示：

```
create table friends "( c1 INT NOT NULL, c2 INT, c3 CHAR(8) )"
```

如果尝试将下列数据记录装入到表中：

```
23, 24, bobby
, 45, john
4,, mary
```

将拒绝第二行，这是因为第一个 INT 是 NULL，但列定义指定了 NOT NULL。包含与 DEL 格式不一致的初始字符的列将生成错误，将拒绝该记录。您可以将此类记录写入转储文件。

在下面这个示例中，将忽略字符定界符外部的 DEL 数据，但会生成警告。例如：

```
22,34,"bob"
24,55,"sam" sdf
```

LOAD 实用程序将在表的第三列中装入"sam"，并且将会在一条警告消息中标记字符"sdf"。不会拒绝该记录。另一个示例：

```
22 3, 34,"bob"
```

LOAD 实用程序将装入 22，34，"bob"并生成一条警告消息，该消息指出忽略了第一列中 22 后面的某些数据。不会拒绝该记录。

例 6-11 (从 CURSOR 装入)

假定源表和目标表位于同一数据库中，并且它们的定义如表 6-5 所示。

表 6-5 源表和目标表的结构

表 DEV.TABLE1 有 3 列：结构如下	表 DEV.TABLE2 有 3 列：结构如下
ONE INT	ONE INT
TWO CHAR(10)	TWO CHAR(10)
THREE DATE	THREE DATE

游标 MYCURSOR 是按以下方式定义的：

```
declare mycursor cursor for select * from dev.table1
```

以下命令将 DEV. TABLE1 中的所有数据装入到 DEV. TABLE2 中：

```
load from mycursor of cursor method P(1,2,3)  insert into  dev.table2(one,two,three)
```

注意：

1. 在单个 LOAD 命令中只指定了一个游标名。即，不允许 load from mycurs1，mycurs2 of cursor...。

2. 对于从游标装入来说，有效的 METHOD 值只有 P 和 N。

3. 在此示例中，由于插入列列表(one，two，three)表示默认值，所以可以将 METHOD P 和该插入列列表省略。

4. DEV. TABLE1 可以是表、视图、别名或昵称。

例 6-12　限制装入行数。

用 ROWCOUNT 选项可以指定从文件开始处装入的记录数。例如，导入前 3 条记录。

```
LOAD FROM D:\STAFF.TXT OF DEL ROWCOUNT 3 INSERT INTO STAFF1
```

注意：

staff. txt 文件是我们从表 staff 中导出的 DEL 格式的文件。

例 6-13　出现警告信息时强令装入操作失败。

在某些情况下，文件中的数据必须全部成功输入到目标表中才算成功，即使有一条记录出错也不行。在这种情况下，可以使用 WARNINGCOUNT 选项。把输入文件 staff. txt 中第一列数据类型为数字并等于 320 的行替换为："abcf"，"aaa"，"sdfg"，执行下列命令：

```
LOAD FROM D:\STAFF.TXT OF DEL WARNINGCOUNT 1 INSERT INTO STAFF1
```

运行结果包含下面的警告：

```
SQL3118W  在行 "32" 列 "1"中的字段值不能转换为 SMALLINT 值，但是目标列不可为空。未装入该行。
SQL3185W  当处理输入文件的第 "32" 行中的数据时发生先前的错误。
SQL3502N  实用程序遇到了"1" 个警告，它超过了允许的最大警告数。
```

此时无法对表 STAFF1 进行操作，例如：

```
SELECT * FROM STAFF1
```

会返回：

```
ID     NAME       DEPT   JOB   YEARS  SALARY    COMM
------ ---------- ------ ----- ------ --------- ---------
SQL0668N  由于表 "USER.STAFF1" 上的原因代码 "3"，所以不允许操作。
SQLSTATE=57016
原因是：表处于“装入挂起”状态。对此表的先前的 LOAD 尝试失败。在重新启动或终止 LOAD 操作之前不允许对表进行存取。
```

解决方法为：通过分别发出带有 RESTART 或 TERMINATER 选项的 LOAD 来重新启动或终止先前失败的对此表的 LOAD 操作。

包含 TERMINATER 的 LOAD 命令可以终止装入进程，使目标表恢复正常可用状态：

```
LOAD FROM D:\STAFF.TXT OF DEL TERMINATE INTO STAFF1
```

包含 RESTART 的 LOAD 命令可以在源文件修改正确的时候使用，使装入进程重新

开始：

```
LOAD FROM D:\STAFF.TXT OF DEL RESTART INTO STAFF1
```

例 6-14 防止产生警告信息。

使用 NOROWWARNINGS 文件类型修饰符可以禁止产生警告信息，当装入过程可能出现大量警告信息，而用户对此又不感兴趣时，可以使用该选项，这样可以大大提高装入的效率。打开 STAFF. TXT 文件，把第一列等于 320 的行替换为："abcf"，"aaa"，"sdfg"。

```
LOAD FROM D:\STAFF.TXT OF DEL MODIFIED BY NOROWWARNINGS INSERT INTO STAFF
```

运行结果中，第 32 行出错，该行无法装入，但是不产生警告信息。

例 6-15 生成统计数据。

使用 STATISTICS 选项可以在装入的过程中生成统计数据，这些统计数据可以供优化器确定最有效的执行 SQL 语句的方式。可以对表和索引产生不同详细程度的统计数据。

对表和索引产生最详细的统计数据：

```
    LOAD FROM D:\STAFF.TXT OF DEL REPLACE INTO STAFF1 STATISTICS YES WITH D
ISTRIBUTION AND DETAILED INDEXES ALL
```

对表和索引都产生简略的统计数据：

```
    LOAD FROM D:\STAFF.TXT OF DEL REPLACE INTO STAFF1 STATISTICS YES AND IN
DEXES ALL
```

6.4.4 在线 LOAD

在 DB2 V8 之前，当一个表被装载时，LOAD 实用程序用一个超级排它锁将它锁定。并且 LOAD 所在的表空间也处于 LOAD PENDING 状态而无法访问。在 DB2 V8 以后为了增强 LOAD 的可用性，我们可以指定 LOAD 的一个选项 ALLOW READ ACCESS。如果没有指定该选项，默认是 ALLOW NO ACCESS，此选项在装载完成之前，不能访问正在装载的数据。这个选项使正在装载数据的表处于 LOAD IN PROGRESS 状态和 READ ACCESS ONLY 状态。ALLOW READ ACCESS 选项导致被装载的表以共享的方式锁定。我们可以访问表中已有的数据，但是不能访问新装载的那部分数据。图 6-14 是在线和离线 LOAD 的示意图。

通过图 6-14 我们可以看到 ALLOW NO ACCESS 选项以独占方式锁定表，在装入该表时不允许对表数据进行访问，这个锁直到 LOAD 完成才释放。

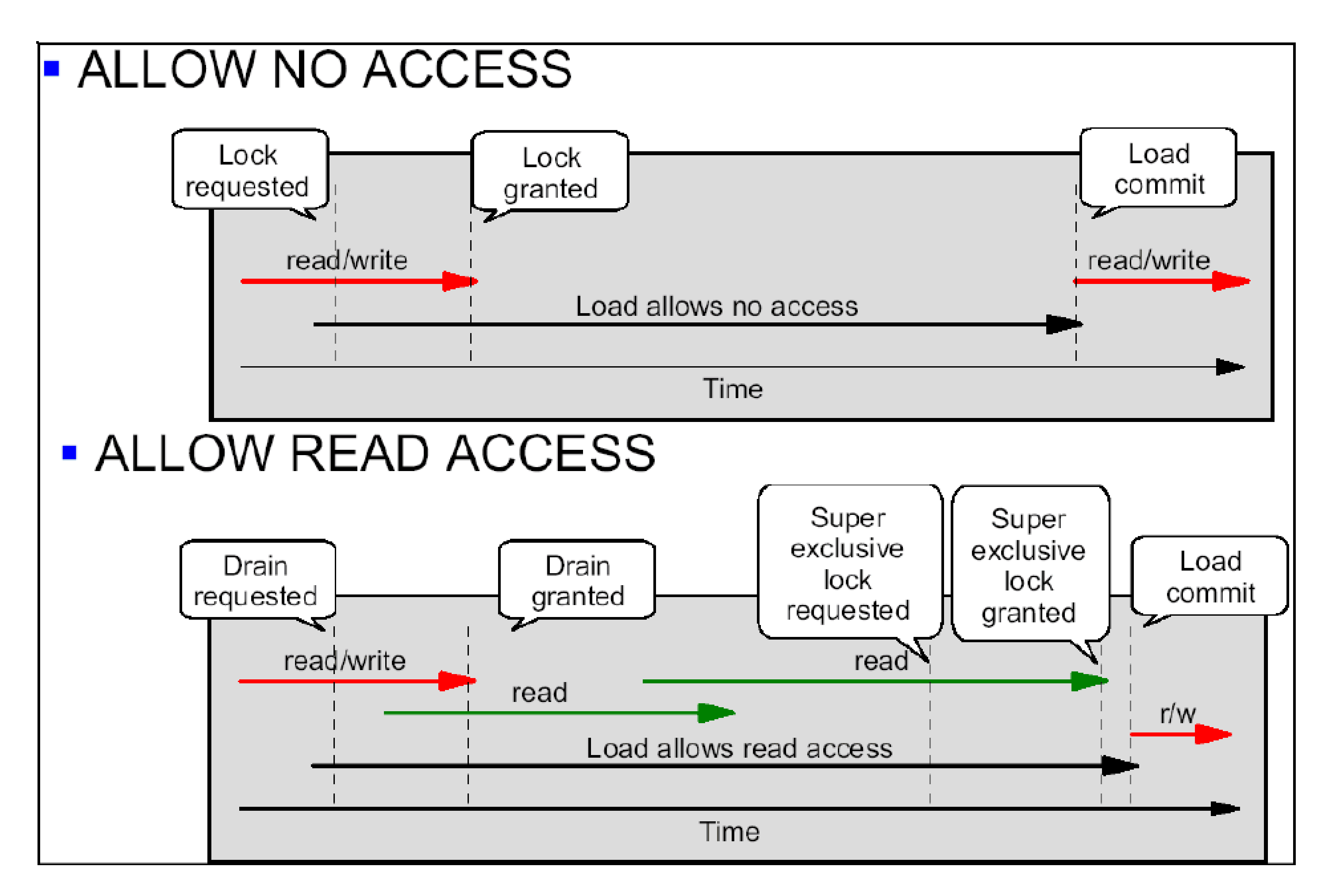


图 6-14　在线和离线 LOAD

对于 ALLOW READ ACCESS 选项我们看到，当我们开始 LOAD 时，如果某个应用程序已经锁住了目标表，那么 LOAD 实用程序就必须等到这些锁被释放。为了不必等到一个锁释放出来，可以使用 LOAD 命令中的 LOCK WITH FORCE 选项，使持有冲突锁的其他应用程序强制离开。ALLOW READ ACCESS 选项不允许其他应用程序对该表进行任何写访问，但允许对预先存在的数据进行读访问。在装入操作的整个执行过程中，除了操作开始和操作结束时以外，都允许进行读访问。下面我们详细讲解 ALLOW READ ACCESS 访问机制。

首先，在设置阶段接近结束时，装入操作将获取特殊的 Z 锁定并占用一小段时间。如果某个应用程序在装入操作请求这个特殊的 Z 锁定前，对该表挂起了不兼容的锁定，那么装入操作在发生超时和失败之前将等待一段有限的时间，以允许这个不兼容的锁定被释放。等待时间长度由 *locktimeout* 数据库配置参数确定。如果指定了 LOCK WITH FORCE 选项，装入操作就会强制其他应用程序释放锁定以避免超时。装入操作获取特殊的 Z 锁定、落实设置阶段、释放该锁定并进入装入阶段。在 ALLOW READ ACCESS 方式的装入操作启动后，任何对该表请求读访问锁定的应用程序都将获得该锁定，而不会与这个特殊的 Z 锁定发生冲突。尝试读取目标表中现有数据的新应用程序也能够成功地完成操作。

其次，在装入操作结束时，LOAD 实用程序在落实数据之前对该表获取互斥锁定(Z 锁定)。LOAD 实用程序将等待所有对该表挂起了锁定的应用程序释放那些锁定。这会导致落实数据前发生延迟。LOCK WITH FORCE 选项用来强制有冲突的应用程序释放锁定，这样

装入操作就能够继续执行而不必等待。通常，ALLOW READ ACCESS 方式的装入操作获取互斥锁定并占用一小段时间；但是，如果指定了 USE <tablespace-name>选项，就会在整个索引复制阶段占用互斥锁定。

在装入操作执行过程中，可以对在装入操作启动前存在的表数据和索引数据进行查询。下面我们举一个例子来说明。

```
 创建包含一个整数列的表：
     create table t1 (id int)
 装入三行：
     load from t1.txt of del insert into t1   ...
     读取的行数                   = 3
     跳过的行数         = 0
     已装入的行数                 = 3
     拒绝的行数         = 0
     删除的行数         = 0
     已落实的行数                 = 3
查询该表：
     select * from t1
                    ID
                 -----------
                     1
                     2
                     3
    3 record(s) selected.
在指定了 ALLOW READ ACCESS 选项的情况下执行装入操作并装入另外两行数据：
     load from t12.txt of del insert into t1 allow read access
同时，使用另一个连接，在装入操作执行过程中查询该表：
     select * from t1
                    ID
                  -----------
                     1
                     2
                     3
    3 record(s) selected.
等待装入操作完成，然后查询该表：
     select * from t1
                    ID
                 -----------
                     1
                     2
                     3
                     4
```

```
                    5
    5 record(s) selected.
```

由于 ALLOW READ ACCESS 选项允许用户在任何时间(甚至在装入操作执行时或者在装入操作失败后)访问表数据，所以，装入大量数据时，此选项非常有用。在 ALLOW READ ACCESS 方式下，装入操作的行为独立于应用程序的隔离级别。即，具有任何隔离级别的应用始终能够读取预先存在的数据，但它们在装入操作完成前无法读取新装入的数据。

对于 ALLOW READ ACCESS 选项，如果重新构建完整的索引，那么将创建索引的一个影子副本。当 LOAD 实用程序进入索引复制阶段(见装载过程的 4 个阶段)时，目标表将离线，新的索引被复制到目标表空间。

无论指定哪种表访问选项，装载操作都需要得到各种不同的锁才能继续。

6.4.5　监控 LOAD 进度

LOAD QUERY

当一个表被装载时，LOAD 实用程序用一个排它锁将它锁定。在装载完成之前，对表的其他访问是不允许的。这是 ALLOW NO ACCESS 选项的默认行为。在那样的装载期间，表处于 LOAD IN PROGRESS 状态。如果我们想查看 LOAD 的进度，可以通过 LOAD QUERY 命令检查装载操作的状态和返回表状态：

```
LOAD QUERY TABLE shcemaname.table_name
```

下面我们举一个使用 LOAD QUERY 查询 LOAD 进度的例子。

```
C:\>db2 load query table oracle.dept1
SQL3109N  实用程序正在开始从文件 "C:\dept.txt" 装入数据。
SQL3500W  在时间 "2008-11-23 00:56:21.772682"，实用程序在开始 "LOAD"。
SQL3519W  开始装入一致点。输入记录数 = "0"。
SQL3520W  “装入一致点”成功。
SQL3110N  实用程序已完成处理。从输入文件读了"229376" 行。
SQL3519W  开始装入一致点。输入记录数 = "229376"。
SQL3520W  “装入一致点”成功。
SQL3515W  在时间 "2008-11-23 00:56:23.415390"，实用程序已经完成了"LOAD"。
SQL3532I  Load 实用程序当前正处于"LOAD" 阶段。
读取行数          = 229376
跳过行数          = 0
装入行数          = 229376
拒绝行数          = 0
删除行数          = 0
落实行数          = 229376
```

```
警告数              = 0
表状态:
  正在装入(load in progress)
C:\>
```

LIST UTILITIES

调用 LOAD 实用程序后，除了使用 LOAD QUERY 查询 LOAD 进度外，我们可以使用 LIST UTILITIES 命令来监视装入操作的进度。在以 INSERT 方式、REPLACE 方式或 RESTART 方式执行装入操作时，将提供详细进度监视支持。发出带有 SHOW DETAILS 选项的 LIST UTILITIES 命令来查看关于当前装入阶段的详细信息。以 TERMINATE 方式执行装入操作时，将无法获取详细信息，LIST UTILITIES 命令将仅仅显示装入终止实用程序当前正在运行，下面我们举一个例使用 LIST UTILITIES 监控 LOAD 进度的例子。

```
C:\>db2 list utilities show detail
标识                                      = 7
类型                                      = LOAD
数据库名称                                = SAMPLE
分区号                                    = 0
描述       = OFFLINE LOAD DEL AUTOMATIC INDEXING INSERT COPY NO ORACLE  .DEPT1
开始时间                = 2008-11-23 01:01:50.423478
状态                    = 执行
调用类型                = 用户
进度监视:
   阶段号                        = 1
      描述                       = SETUP
      总计工作                   = 0 bytes
      已完成的工作               = 0 bytes
      开始时间                   = 2008-11-23 01:01:50.423489
   阶段号 [当前]                  = 2
      描述                                    = LOAD(构建阶段)
      总计工作                   = 226508 rows
      已完成的工作               = 170081 rows
      开始时间                   = 2008-11-23 01:01:50.524716
```

6.4.6 LOAD 期间和之后的表空间状态

在装入操作期间，LOAD 实用程序使用表空间状态来保持数据库一致性。这些状态通过控制对数据的访问或引发用户操作来起作用。

可以使用 LIST TABLESPACES、LOAD QUERY 和 LIST UTILITIES 命令来检查表空间状态。表空间可以同时处于多种状态。关于每种表空间状态的详细例子，在本书第 15

章有详细的总结，本节限于篇幅就不展开了。LOAD 期间和之后的常见状态有：

正常(normal)

“正常”状态是创建表空间后该表空间的初始状态，它指示当前没有(异常)状态影响表空间。

只读访问

如果指定了 ALLOW READ ACCESS 选项，那么表将处于“只读访问”状态。在调用 LOAD 命令前存在的表数据在装入操作运行期间可供只读访问。如果指定了 ALLOW READ ACCESS 选项并且装入操作失败，那么在装入操作前存在的表数据在故障发生后将继续可供只读访问。

正在装入(load in progress)

“正在装入”状态指示正在表空间上进行装入。此状态不允许在装入操作期间备份从属表。“正在装入”表空间状态与“正在装入”表状态(所有装入操作中都使用此状态)不同，因为仅当对可恢复数据库指定了 COPY NO 参数时，LOAD 实用程序才使表空间处于“正在装入”状态。

表空间在装入操作处于多种状态的表的示例

将包含大量数据的输入文件(staffdata.del)装入到 NEWSTAFF 表中，如下所示：

```
connect to sample;
create table newstaff like staff;
load from staffdata.del of del insert into newstaff allow read access;
```

并且打开另一个会话并发出下列命令：

```
connect to sample;
load query table newstaff;
```

LOAD QUERY 命令将显示 NEWSTAFF 表处于“只读访问”和“正在装入”状态：

```
表状态：
  正在装入(load in progress)
  只读访问(READ ACCESS)
```

备份暂挂(backup pending)

如果对可恢复数据库执行装入操作并且指定 COPY NO 参数，那么在第一次落实后表空间将处于“备份暂挂”表空间状态。不能更新处于“备份暂挂”状态的表空间。通过备

份表空间即可使表空间脱离“备份暂挂”状态。由于装入操作开始时会更改表空间状态并且不能回滚，所以即使取消装入操作，表空间也保持处于“备份暂挂”状态。

表空间状态的示例

将输入文件(staffdata.del)装入到 NEWSTAFF 表中，如下所示：

```
update db cfg for sample using logretain recovery;---更改为归档日志，我们下一章讲解
backup db sample;
connect to sample;
create table newstaff like staff;
load from staffdata.del of del insert into newstaff copy no;
```

并且打开另一个会话并发出下列命令：

```
connect to sample;
list tablespaces;
```

那么 USERSPACE1(样本数据库的默认表空间)将处于“正在装入”状态，并且在第一次提交后，将处于“备份暂挂”状态。在装入操作完成后，LIST TABLESPACES 命令表明 USERSPACE1 现在处于“备份暂挂”(backup pending)状态：

```
表空间标识                    = 3
名称                          = IBMDB2SAMPLEREL
类型                          =数据库管理空间
内容                          = 所有持久数据。大型表空间。
状态                          = 0x0020
  详细解释：
    备份暂挂(backup pending)
```

复原暂挂(restore pending)

如果使用 COPY NO 选项成功执行了装入操作、复原数据库，然后前滚该操作，那么相关表空间将处于“复原暂挂”状态。要使表空间脱离“复原暂挂”状态，必须执行复原操作。

装入暂挂(load pending)

“装入暂挂”表状态指示装入操作失败或被中断。可以执行以下其中一种方法来除去“装入暂挂”状态：

- 找出故障原因。例如，如果 LOAD 实用程序耗尽了磁盘空间，那么对表空间添加容器。然后，重新启动(restart)装入操作。

- 终止(terminate)装入操作。
- 对装入操作失败时所处理的那个表进行 LOAD REPLACE 操作。
- 使用最新的表空间或数据库备份，通过 RESTORE DATABASE 命令恢复所装入的表的表空间，然后执行进一步的恢复操作。

不可重新启动装入

处于“不可重新启动装入”状态时，表已部分装入，并且不允许装入重新启动操作。在下面两种情况下，表会处于“不可重新启动装入”状态：

- 在未能成功地重新启动或终止的失败装入操作后，执行前滚操作；
- 根据表处于“正在装入”或“装入暂挂”状态时创建的联机备份执行复原操作。

表还将处于“装入暂挂”状态。要使表脱离“不可重新启动装入”状态，发出 LOAD TERMINATE 或 LOAD REPLACE 命令。

设置完整性暂挂

“设置完整性暂挂”状态指示已装入的表有未经验证的约束。当 LOAD 实用程序开始对带有约束的表执行装入操作时，它就会使该表处于此状态。使用 SET INTEGRITY 语句以使该表脱离“设置完整性暂挂”状态。下面我们举一个设置完整性暂挂的例子：

```
1. Connect to sample                          连接到 SAMPLE 数据库
2. CREATE TABLE STAFF1 LIKE STAFF             创建一个结构与 staff 表相同的表：
3. alter table staff1 add constraint chk check(dept<100) 给该表添加一个检查约束
4. LOAD FROM D:\STAFF.TXT OF DEL INSERT INTO STAFF1，打开 STAFF.TXT 文件，把最后一行数据的第三列改为 150，这样这条数据就不满足第 3 步加上的检查约束条件了，然后用 Load 命令从文件中装入数据到 staff1 表中
5. Select * from staff1
此时运行查询命令，会得到错误信息：SQL0668N  由于表 "USER.STAFF1" 上的原因代码 "1"，所以不允许操作。SQLSTATE=57016 原因是装入时有数据违反了检查约束，造成表处于检查挂起状态。
6. set integrity for staff1 check immediate unchecked 解除表的检查挂起状态，使用：再次运行查询命令
Select * from staff1
发现表可以正常使用了，其中的违反检查规则的数据也存在。
```

不可用

通过不可恢复的装入操作执行前滚将使表处于“不可用”状态。处于此状态时，表不可用；必须 drop 该表或通过备份复原表。

6.4.7 使用 CURSOR 文件类型来移动数据

通过在使用 LOAD 命令时指定 CURSOR 文件类型，可以将 SQL 查询结果直接装入到目标表中，而不必创建中间导出的文件。

有 3 种方法可用于通过使用 CURSOR 文件类型来移动数据。第一种方法是使用命令行处理器(CLP)，第二种方法是使用 API，第三种方法是使用 ADMIN_CMD 过程。要从 CLP 中执行 LOAD FROM CURSOR 操作，首先必须对 SQL 查询声明游标。一旦声明了游标，就可以发出 LOAD 命令，并将所声明游标的名称用作 *cursorname*，将 CURSOR 用作文件类型。

例 6-16 假定源表和目标表位于同一数据库中，并且它们的定义如表 6-6 所示。表上没有索引，现在我们要把表 T1 的数据导入到表 T2 中，数据量大概为 6 千万条记录。图 6-15 对 CUSOR 方式和其他方式进行了比较。

表 6-6 源表和目标表的结构

表 DEV.T1 有 3 列：结构如下	表 DEV.T2 有 3 列：结构如下
ONE INT	ONE INT
TWO CHAR(10)	TWO CHAR(10)
THREE DATE	THREE DATE

Operation(s)	Elapsed Time
INSERT INTO T2 SELECT * FROM T1 (with logging)	1838 seconds
INSERT INTO T2 SELECT * FROM T1 (using not logged initially)	1818 seconds
EXPORT TO <file> OF IXF SELECT * FROM T1 LOAD FROM <file> OF IXF INSERT INTO T2	99 seconds (31s + 68s)
EXPORT TO <pipe> OF IXF SELECT * FROM T1 LOAD FROM <pipe> OF IXF INSERT INTO T2 (concurrently)	68 seconds
DECLARE CURSOR1 CURSOR FOR SELECT * FROM T1 LOAD FROM CURSOR1 OF CURSOR INSERT INTO T2	68 seconds
EXPORT TO <file> OF IXF SELECT * FROM T1 IMPORT FROM <file> OF IXF INSERT INTO T2	1940 seconds (32s + 1908s)

图 6-15 CUSOR 和其他方式的比较

执行下列 CLP 命令会将 DEV. T1 中的所有数据装入到 DEV. T2 中：

```
c:>db2 +c  DECLARE mycurs CURSOR FOR SELECT TWO, ONE, THREE FROM dev.t1
```

```
c:>db2 +c  LOAD FROM mycurs OF cursor INSERT INTO dev.t2  ---+c 是关闭自动提交
```

从图 6-15 中我们可以看出通过游标方式装载数据是花费时间最小的。过去没有 CURSOR 方式时，我们需要把数据落地(图中显示落地花了 31s)，然后再装载。而现在可以把这个落地步骤省掉从而提高 LOAD 速度。

6.4.8　提高 LOAD 性能

在众多数据库中，LOAD 装载速度最快。但是我们还是可以使用各种命令参数来优化装入性能。还有许多对装入来说唯一的文件类型修饰符，在某些情况下，这些修饰符可以极大地提高 LOAD 实用程序的性能。

1. 并行性和装入

LOAD 实用程序使用多个处理器或多个存储设备的硬件配置，如对称多处理器(SMP)环境。

通过使用 LOAD 实用程序，可以有多种方法来并行处理大量数据。一种方法是使用多个存储设备，这允许在装入操作期间利用 I/O 并行性(图 6-16 所示)。另一种方法涉及在 SMP 环境中使用多个处理器，这允许利用分区内并行性(图 6-17 所示)。两种方法可一起使用以提高数据装入速度。

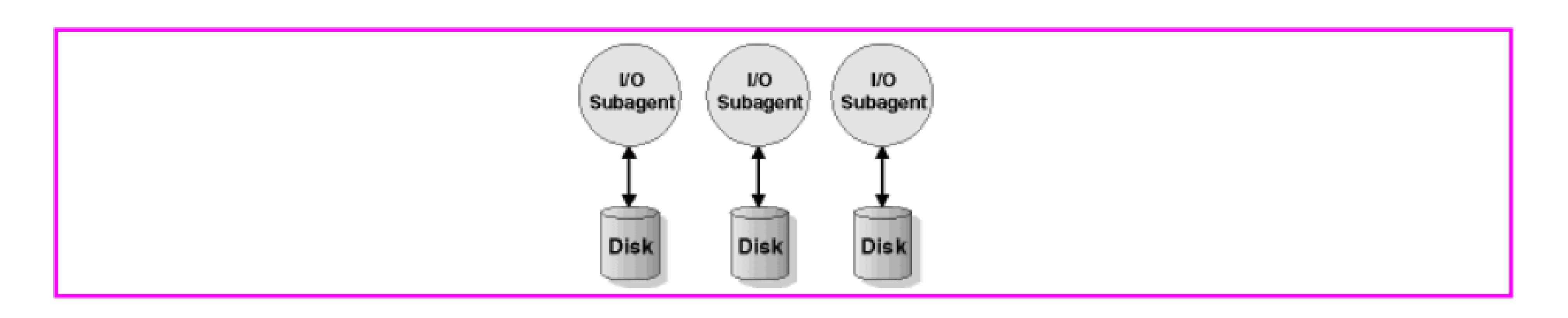


图 6-16　在装入数据时利用 I/O 并行性

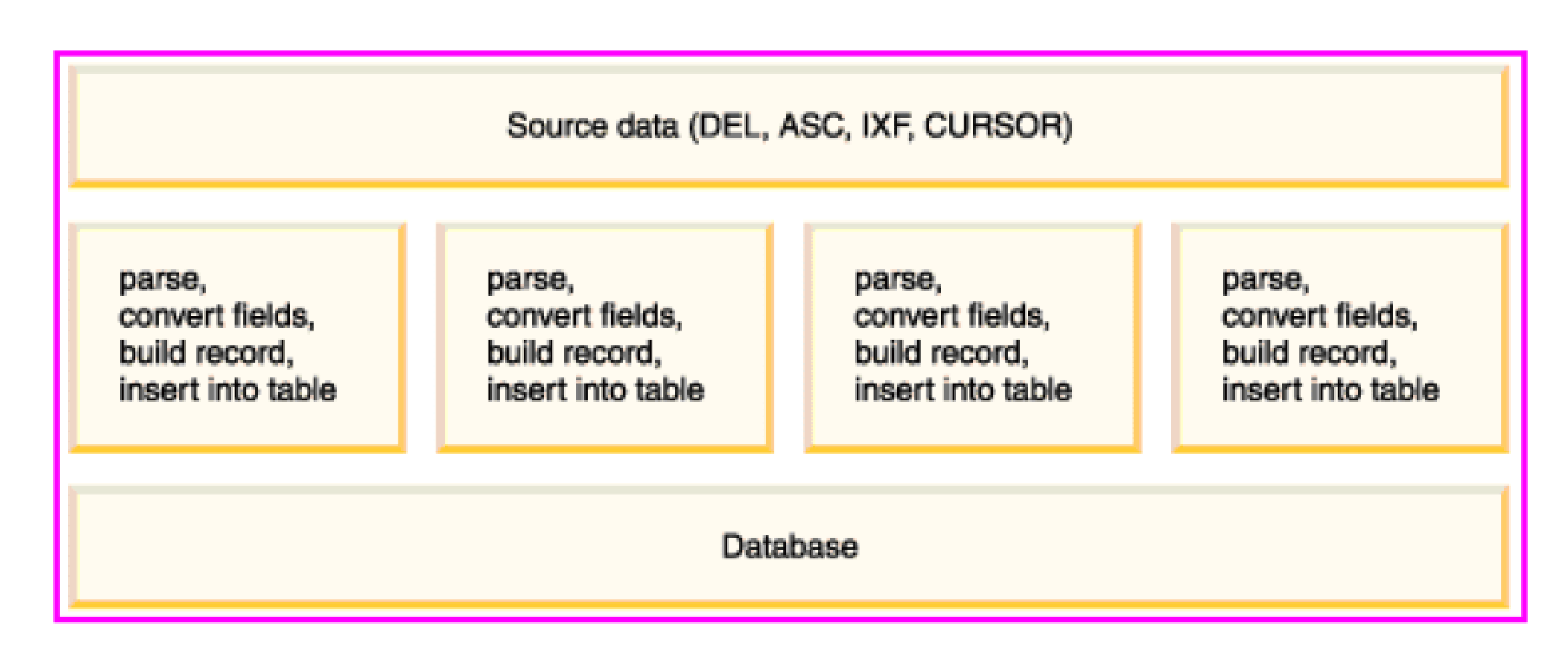


图 6-17　在装入数据时利用分区内并行性

2. 用于提高装入性能的选项

如果用户未指定 DISK_PARALLELISM、CPU_PARALLELISM 和 DATA BUFFER 参数的值，那么 LOAD 实用程序将尝试通过确定这些参数的最优值来获得最好的性能。根据实用程序堆大小及可用空间来进行优化。在尝试调整这些参数以满足特殊需要之前，考虑使用自主 DISK_PARALLELISM 和 CPU_PARALLELISM 设置。

以下是有关 LOAD 实用程序提供的各种选项所带来的性能影响的信息：

- ALLOW READ ACCESS

此选项允许您在进行装入操作时查询表。只能查看装入操作之前表中已存在的数据。如果还指定了 INDEXING MODE INCREMENTAL 选项，并且装入操作失败，那么后续装入终止操作可能必须校正索引中的不一致。这需要涉及大量 I/O 的索引扫描。如果还对装入终止操作指定了 ALLOW READ ACCESS 选项，那么会将缓冲池用于 I/O。这个选项会间接地影响 LOAD 的时间，因为在线 LOAD 会延长装载时间。

- COPY YES 或 COPY NO

使用此参数来指定在装入操作期间是否创建输入数据的副本。仅当启用了前滚恢复时，COPY YES 才适用，并且由于装入操作期间会复制所有装入数据，所以使用此参数会降低装入性能。I/O 活动增加可能会导致 I/O 绑定系统上的装入时间增加。如果指定多个设备或不同磁盘上的多个目录，那么可能会因为此操作而使性能受到影响。仅当启用了前滚恢复时，COPY NO 才适用，并且它不会影响装入性能。但是，所有与已装入的表相关的表空间将处于“备份暂挂”(back pending)状态，并且必须先备份这些表空间才能访问该表。

- CPU_PARALLELISM

借助此参数来使用分区内并行性(如果机器具有此功能)，从而大幅改进装入性能。该参数指定 LOAD 实用程序用于分析、转换、格式化数据记录的进程或线程的数目。如果内存不足以支持指定值，那么实用程序将调整该值。如果未指定此参数，那么 LOAD 实用程序将根据系统上的 CPU 数目选择默认值。

只要满足下列条件，无论此参数的值如何，都将保留源数据中的记录顺序(请参阅图 6-18)：

- ◊ 未指定 anyorder 文件类型修饰符
- ◊ 未指定 PARTITIONING_DBPARTNUMS 选项(并且将多个分区用于分区)
- ◊ 如果表包括 LOB 或 LONG VARCHAR 数据，那么 CPU_PARALLELISM 将设置为 1。在这种情况下不支持并行性。

尽管此参数并未限制为只能供对称多处理器(SMP)硬件使用，但在非 SMP 环境中使用它在性能方面也没什么太大益处。

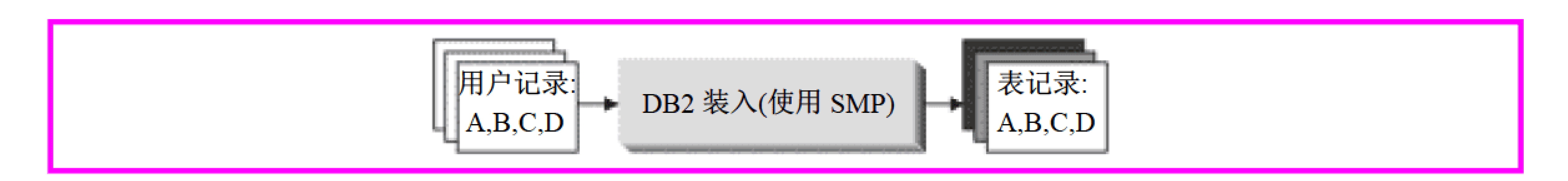


图 6-18　在装入操作期间使用分区内并行性时，将会保留源数据中的记录顺序

- DATA BUFFER

DATA BUFFER 参数指定分配给 LOAD 实用程序以用作缓冲区的内存总量(以 4KB 为单位)。建议此缓冲区在大小上等于若干扩展数据块。数据缓冲区将从实用程序堆分配。根据系统上可用的存储量，应考虑分配更多内存以供 DB2 实用程序使用。可相应修改数据库配置参数 util_heap_sz(实用程序堆大小)。util_heap_sz 的默认值为 5000 4KB 的页。因为 LOAD 实用程序只是使用实用程序堆内存的若干实用程序的其中一个，所以建议将此参数定义的不超过 50%的页供 LOAD 实用程序使用，并且将实用程序堆定义得足够大。

- DISK_PARALLELISM

DISK_PARALLELISM 参数指定 LOAD 实用程序用来将数据记录写至磁盘的进程或线程数。借助此参数在装入数据时使用可用容器，从而大幅改进装入性能。允许的最大数目是 CPU_PARALLELISM 值(LOAD 实用程序使用的实际量)的 4 倍或 50 中较大的数字。默认情况下，DISK_PARALLELISM 等于包含对其装入表的对象的所有表空间中的表空间容器的总和，但当此值超过允许的最大值时除外。

- SAVECOUNT

使用此参数来设置在装入操作的装入阶段建立一致点的时间间隔。为建立一致点而执行活动同步需要花一些时间。如果进行得太频繁，装入性能会大幅下降。如果要装入大量行，那么建议您指定较大的 SAVECOUNT 值(例如，在涉及 1 亿条记录的装入操作中指定值 10 000 000)。只要装入重新启动操作(restart)从装入阶段恢复，该操作就会从上一个一致点自动继续。

- STATISTICS USE PROFILE

收集表统计信息概要文件中指定的统计信息。即使装入操作本身的性能下降(特别是在指定 DETAILED INDEXES ALL 时)，与在完成装入操作后调用 RUNSTATS 实用程序相比，使用此参数来收集数据分发和索引统计信息更有效。为优化性能，应用程序需要尽可能最佳的数据分发和索引统计信息。一旦更新统计信息，应用程序就可以根据最新的统计信息使用新的表数据访问路径。可通过使用 REBIND 命令重新绑定应用程序包来创建新的表访问路径。通过运行带有 SET PROFILE 选项的 RUNSTATS 命令来创建表统计信息概要文件。

将数据装入到大表中时，建议对 stat_heap_sz(统计信息堆大小)数据库配置参数指定较大的值。

- USE <tablespace-name>

如果正在执行 ALLOW READ ACCESS 装入并且建立索引的方式为 REBUILD，那么此参数允许在系统临时表空间中重建索引，并在装入操作的索引复制阶段将其复制回索引表空间。

默认情况下，将在原始索引所在的表空间中构建完全重建的索引(也称为影子索引)。因为原始索引和影子索引同时位于同一表空间中，所以这可能会导致资源问题。如果影子索引与原始索引是在同一个表空间中构建的，那么影子索引将瞬时替换原始索引。但是，如果影子索引是在系统临时表空间中构建的，那么装入操作需要索引复制阶段，该阶段会将索引从系统临时表空间复制至索引表空间。复制阶段将涉及相当多的 I/O。如果其中任一表空间是 DMS 表空间，那么系统临时表空间的 I/O 可能不是顺序进行的。在索引复制阶段将使用 DISK_PARALLELISM 选项指定的值。

3. 文件类型修饰符

文件类型修饰符是用 MODIFIED BY 子句指定的。下面是一些对性能会有影响的文件类型修饰符：

- ANYORDER

默认情况下，LOAD 实用程序将保留源数据的记录顺序。在 SMP 环境中进行装入时，要求并行处理之间保持同步以确保保留该顺序。

在 SMP 环境中，指定 ANYORDER 文件类型修饰符将指示 LOAD 实用程序不保留顺序，由于这样一来就不必执行保留该顺序所需的同步，所以将会提高性能。但是，如果要装入的数据进行了预先排序，那么 ANYORDER 可能会破坏预先排好的顺序，使得后续查询也无法受益于预先排序。

注意：

对于外界来源的数据在使用时，如果 CPU_PARALLELISM 为 1，那么 ANYORDER 文件类型修饰符不起作用，并且它与 SAVECOUNT 选项不兼容。

- BINARYNUMERICS、ZONEDDECIMAL 和 PACKEDDECIMAL

对于固定长度的非定界 ASCII(ASC)源数据，用二进制表示数字数据可能会提高装入时的性能。如果指定了 PACKEDDECIMAL 文件类型修饰符，那么 LOAD 实用程序会使用压缩十进制格式(每个字节占两位)表示十进制数据。如果指定了 ZONEDDECIMAL 文件类型修饰符，那么 LOAD 实用程序会使用分区十进制格式(每个字节占一位)表示十进制数据。对于所有其他数字类型，如果指定了 BINARYNUMERICS 文件类型修饰符，那么 LOAD 实用程序会使用二进制格式表示数据。

- FASTPARSE

使用时务必小心谨慎。如果知道要装入的数据有效，那么没必要让装入像对较可疑的数据那样执行那么多的语法检查。事实上，缩小语法检查的范围可以将装入性能提高大约 10%或 20%。这可以通过使用 FASTPARSE 文件类型修饰符来实现，该修饰符可以减少对 ASC 和 DEL 文件中用户提供的列值执行的数据检查。

- PAGEFREESPACE、INDEXFREESPACE 和 TOTALFREESPACE

随着时间的推移，表中插入和更新的数据不断增加，重组表和索引的需求也就更迫切。一种解决方案是使用 PAGEFREESPACE、INDEXFREESPACE 和 TOTALFREESPACE 来增大用于表和索引的可用空间量。前两个修饰符优先于 PCTFREE 值，它们指定要作为可用空间保留的数据和索引页数的百分比，而 TOTALFREESPACE 指定要作为可用空间追加至表的总页数的百分比。

4. LOAD 性能总结

图 6-19 是使用上述命令选项和文件修饰符前后的性能比较。

文件格式 IXF性能最好，建议多用IXF	使用的LOAD命令选型和Modified by文件修饰符对性能的影响。因为硬件环境不一样，读者应主要关心使用这些影响LOAD性能选项后的性能相对提高度	GB/h/CPU	Rows/Sec	Improvement
IXF	ANYORDER, DATA BUFFER 40000, CPU_PARALLELISM 5, DISK_PARALLELISM 8	50.8±0.2	127900±300	18%
ASC	ANYORDER, FASTPARSE, DATA BUFFER 40000, CPU_PARALLELISM 6, DISK_PARALLELISM 8	29.2±0.1	57300±200	36%
DEL	ANYORDER, FASTPARSE, DATA BUFFER 40000, CPU_PARALLELISM 6, DISK_PARALLELISM 8	28.0±0.1	54800±60	29%
CURSOR	ANYORDER, DATA BUFFER 40000, CPU_PARALLELISM 2, DISK_PARALLELISM 8, INTRA_PARALLEL ON, DFT_DEGREE 18	14.0±0.1	50200±300	14%

图 6-19　使用命令选项和文件修饰符前后的性能比较

从图 6-19 中我们可以看到，使用 IXF 格式的性能比 DEL 和 ASC 要高，所以建议大家在数据移动时最好使用 IXF 格式。同时因为硬件环境的差异，大家只需要关注使用这些选项的相对提高度，而不必关注特定的 LOAD 时间。

6.4.9 LOAD 失败恢复

DB2 将在装入处理期间创建临时二进制文件。这些文件用于装入崩溃恢复、装入终止操作、警告和错误消息以及运行时控制数据。装入临时文件将在装入操作完成而未发生任何错误时自动清除。临时文件将写至通过 LOAD 命令的 temp-pathname 参数指定的路径。默认路径为数据库目录的子目录。

临时文件路径在服务器上，并且由 DB2 实例以独占方式访问。因此，对 temp-pathname 参数指定的任何路径名限定都必须反映服务器(而不是客户机)的目录结构，并且 DB2 实例所有者对该路径必须具有读写许可权。图 6-20 显示了一个 LOAD 异常终止产生的临时文件。

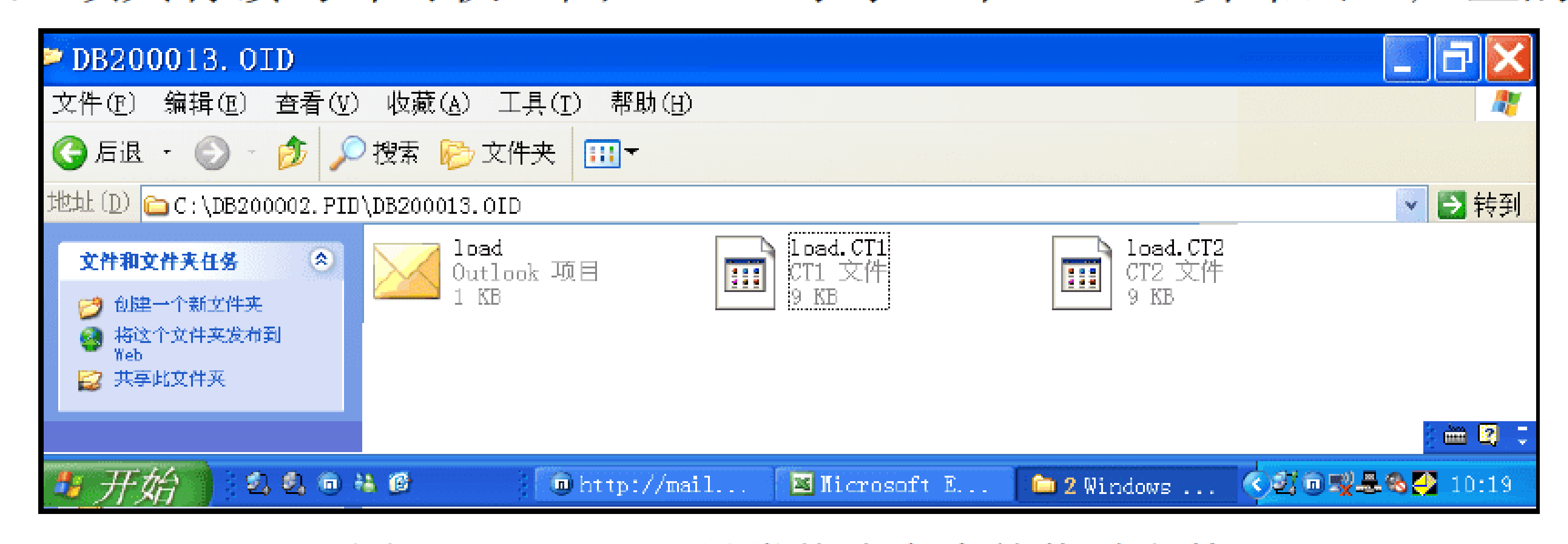


图 6-20 LOAD 异常终止产生的临时文件

当 LOAD 正常结束时，临时文件自动删除。临时文件主要用于异常恢复。

注意：

写至此路径的临时文件在任何情况下都不能编辑修改。编辑修改这些临时文件将导致装入操作失败，并且会使数据库陷入危险状况。

解决导致装入操作失败的情况后，重新发出 LOAD 命令。确保指定的参数与原始命令中的参数完全相同，以便 LOAD 实用程序可以找到必需的临时文件终止该操作、重新装入表或重新启动装入操作。如果要禁止读访问，那么不必指定完全相同的参数。还可以将指定了 ALLOW READ ACCESS 选项的装入操作作为 ALLOW NO ACCESS 选项重新启动。

如果 LOAD 实用程序因为用户错误(例如，数据文件不存在或列名无效)而不能启动，那么操作将终止并让目标表处于正常状态。

装入操作开始时，目标表将处于“正在装入”状态。出现故障时，表状态将更改为“装入暂挂”。要使表脱离该状态，可以发出 LOAD TERMINATE 以回滚操作，发出 LOAD REPLACE 以重新装入整个表，或者发出 LOAD RESTART。

通常，在这种情况下，最好重新启动装入操作。由于 LOAD 实用程序是从装入操作最后成功到达的位置而不是从该操作的开头重新启动装入操作，所以这样做可以节省时间。操作重新启动的准确位置取决于在原始命令中指定的参数。如果指定了 SAVECOUNT 选

项，并且上一个装入操作在装入阶段失败，那么装入操作将在它到达的最后一个一致点重新启动。否则，装入操作在成功到达的最后一个阶段(装入、构建或删除阶段)开始时重新启动。

如果要装入 XML 文档，那么该行为稍有不同。因为在装入 XML 数据时不支持 SAVECOUNT 选项，所以在装入阶段失败的装入操作将从操作的起始处重新启动。正如其他数据类型一样，如果在构建阶段装入失败，那么将在 REBUILD 方式下构建索引，因此会扫描该表以便从每一行获取所有索引键；但是，也必须扫描每个 XML 文档以获取索引键。扫描 XML 文档以查找索引键的这一过程要求对它们重新进行语法分析，这是成本高昂的操作。而且，诸如区域和路径索引之类的内部 XML 索引需要先重构，这也要求扫描 XDA 对象。

如果下列命令产生的装入操作失败：

```
LOAD FROM file name OF file type SAVECOUNT n
MESSAGES message_file  load_method INTO target_tablename
```

您可以通过将指定的装入方法(load_method)替换为 RESTART 方法来重新启动该操作：

```
LOAD FROM file_name OF file_type  SAVECOUNT n
MESSAGES message_file  RESTART   INTO target_tablename
```

不能重新启动(restart)的失败装入

如果失败或中断的装入操作中使用的表处于“不可重新启动装入”状态，那么不能重新启动该操作。不能重新启动的原因如下：

- 在未成功地重新启动或终止的失败装入操作后执行了前滚操作
- 根据表处于“正在装入”或“装入暂挂”状态时创建的联机备份执行了复原操作

应该发出 LOAD TERMINATE 或 LOAD REPLACE 命令。

重新启动或终止 ALLOW READ ACCESS 装入操作

如果以 ALLOW READ ACCESS 方式的装入操作在装入(load)阶段被中断或取消，那么可以使用 ALLOW READ ACCESS 选项重新启动(restart)或终止(terminate)指定了 ALLOW READ ACCESS 选项的已中断或已取消的装入操作。它将在装入阶段重新启动。如果它在装入阶段以外的任何阶段(build、delete 和 index copy)被中断或取消，那么它将在构建(build)阶段重新启动。但是如果索引对象不可用或标记为无效，那么不允许 ALLOW READ ACCESS 方式的装入重新启动。如果原始装入操作在索引复制阶段被中断或取消，那么因为索引可能已损坏而不允许 ALLOW READ ACCESS 方式的重新启动操作。发出 LOAD TERMINATE 命令通常会导致已中断或已取消的装入操作以最短延迟回滚。但是，对指定了 ALLOW READ ACCESS 和 INDEXING MODE INCREMENTAL 的装入操作发出 LOAD

TERMINATE 命令时，LOAD 实用程序扫描索引和校正任何不一致时会有延迟。此延迟的长度取决于索引的大小，并且无论是否对装入终止操作指定 ALLOW READ ACCESS 选项，都会发生延迟。如果原始装入操作在到达构建阶段前失败，那么不会发生延迟。

如果以 ALLOW NO ACCESS 方式执行装入操作，那么当原始装入操作到达构建阶段并且索引有效时，将在删除(delete)阶段发生重新启动操作。如果索引标记为无效，那么 LOAD 实用程序将从构建阶段重新启动装入操作。

6.4.10 LOAD 和 IMPORT 比较

前面我们讲解了 IMPORT 和 LOAD，表 6-7 对 LOAD 和 IMPORT 进行了详细比较。

表 6-7 LOAD 和 IMPORT 的详细比较

IMPORT 实用程序	LOAD 实用程序
移动大量数据时速度较慢。本质是 SQL 操作，导入期间数据库写日志、检查约束，触发器，面向 row，所以速度比较慢	移动大量数据时，由于 LOAD 实用程序将格式化的页直接写入数据库，所以速度比 IMPORT 实用程序快
限制使用分区内并行性。只能通过在 ALLOW WRITE ACCESS 方式下并发调用 import 实用程序来实现分区内并行性	可使用分区内并行性。通常，这需要对称多处理器(SMP)环境。支持 CPU、DISK 和 FETCH 并行性
数据源是平面文件	数据源可以为平面文件、磁带和命名管道
不支持 FASTPARSE	支持 FASTPARSE，减少了对用户提供的数据进行的数据检查工作
索引不重建	索引重建，可以选择索引重建方式
能够创建 PC/IXF 格式的表、层次表和索引	表和索引必须存在
不支持导入到具体化查询表中	支持装入到具体化查询表中
ASC、DEL、WSF 和 IXF 文件格式	ASC、DEL、IXF 和 CURSOR
不支持 BINARYNUMERICS、PACKEDDECIMAL、ZONEDDECIMAL	支持 BINARYNUMERICS、PACKEDDECIMAL、ZONEDDECIMAL
无法覆盖定义为 GENERATED ALWAYS 的列	通过使用 GENERATEDOVERRIDE 和 IDENTITYOVERRIDE 文件类型修饰符，可以覆盖 GENERATED ALWAYS 列

(续表)

IMPORT 实用程序	LOAD 实用程序
支持导入到表、视图和昵称中	仅支持装入到表中
所有行都进行日志记录	执行最少的日志记录
导入期间触发器工作	不支持触发器
如果导入操作被中断，并且指定了 COMMITCOUNT，那么该表可供使用，并且将包含最后一次落实(COMMIT)之前已装入的行。用户可以重新启动导入操作，也可以按原样接受该表	如果装入操作被中断，并且指定了 SAVECOUNT，那么在重新启动装入操作、调用装入终止操作或者根据尝试执行装入操作前创建的备份映像复原表空间之前，该表将保持装入暂挂状态并且不可用
所需空间量大概等于最大索引大小加上 10%。此空间是从数据库中的临时表空间中获取的	所需空间量大概等于对该表定义的所有索引的大小之和，并且可能是此大小的两倍。此空间是从数据库中的临时空间中获取的
在导入操作期间将验证所有约束	LOAD 实用程序将检查唯一性并计算生成列值，但必须使用 SET INTEGRITY 来检查所有其他约束
在导入操作期间，将逐个地把键值插入到索引中	对键值进行排序，装入数据后构建索引
如果需要更新的统计信息，导入操作完成后必须运行 RUNSTATS 实用程序	如果正在替换表中的所有数据，那么可以在装入操作执行期间收集统计信息
导入文件必须在调用 IMPORT 实用程序的客户机上	根据指定的选项，装入文件或管道可以在包含数据库的数据库分区上，也可以在所连接的调用 LOAD 实用程序的远程客户机上 注：只能从服务器端读取 LOB 和 XML 数据
不需要备份映像。由于 IMPORT 实用程序使用 SQL 插入，所以将记录活动，因此发生故障时不需要备份就可以恢复这些操作	在装入操作执行期间，指定 COPY YES，可以自动创建表空间备份映像

6.5 数据移动性能问题

如果我们希望提高数据移动的速度，那么对于 IMPORT 和 EXPORT 来说，它们的本质是执行 SQL 语句，所以设置大的缓冲池、util_heap_sz 和排序堆(sortheap)会提高它们的速度。对于 LOAD 来说，我们在 6.4.8 节已经讲解。这里我们再做一个回顾。

compound x

指定 DB2 IMPORT 实用工具每次插入一块(x 行)数据，而非每次插入一行。这可以导致性能的提高。x 的值可以是 1 至 100 之间的任意整数(包含 1 和 100 在内)。导入实用工具使用非原子复合 SQL 来插入数据：不管是否有错，都会尝试所有的插入。使用该选项大概可以有 30%左右的性能提升。

```
    create table emptemp like employee;
    export to empdata.ixf of ixf messages export.msg select * from employee;
    import from empdata.ixf of ixf modified by compound=100 messages import.msg
insert into emptemp;
```

data buffer x

指定在执行 LOAD 的时候使用的数据缓冲的大小，单位为 4KB。在加载大量数据的时候将此值设置较大能够极大地提高数据加载的性能。这部分内存将会从数据库的 util_heap_sz 指定的内存中分配。所以如果需要较大的 data buffer 则需要同时增加数据库参数 util_heap_sz 的大小。如果不指定此选项，LOAD 会在启动的时候自动地以空闲的 util_heap_sz 中的一定比例来分配内存。

使用方法如下：

```
    load from test.del of del insert into tbname data buffer 2000
```

CPU_PARALLELISM x

指定在运行 LOAD 时，为了解析、转换和格式化而创建的进程数或线程数。从而可以利用分区内并行性来提高性能。如果加载的数据是经过排序的，则此选项特别有用。如果此参数为 0 或不指定此参数，则 LOAD 会自动地设置一个值，如当前可用的 CPU 的数量。

语法如下：

```
    load from test.del of del insert into tbname CPU_PARALLELISM 10
```

DISK_ PARALLELISM x

指定在运行 LOAD 时，为了将数据写入表空间的容器中而创建的进程数或线程数。如果目标表空间中存在多个容器，那么此选项可以极大地提高 I/O 性能。

语法如下：

```
load from test.del of del insert into tbname DISK_PARALLELISM 10
```

除了 IMPORT、EXPORT 和 LOAD 工具的文件修饰符外，数据库层面的 bufferpool、util_heap_sz 和 sortheap 的大小对数据移动也至关重要。

6.6　DB2MOVE 和 DB2LOOK

DB2 有一对非常有用的工具，可以帮助您实现这种跨平台的数据移动。DB2MOVE 是用于在 DB2 数据库之间移动大量表的一个数据移动工具。DB2MOVE 利用了 DB2 的数据移动工具(EXPORT、IMPORT、LOAD 和 COPY)来移动数据库表。然而，由于数据库的内容远远不止用户表，因此您需要使用其他方法在不同的数据库之间迁移其他数据库对象，例如约束、触发器、索引、序列、表空间、缓冲池等。这就是 DB2LOOK 工具出现的原因。使用这个工具，您可以在源数据库中捕获到定义这些对象的数据定义语言(DDL)，并在目标数据库中使用这些数据定义语言重新创建这些对象。

6.6.1　数据库移动工具——DB2MOVE

DB2MOVE 工具将一组用户表从系统编目表中提取出来，并将每个表以 PC/IXF 格式导出。然后，PC/IXF 文件可以被导入或装载到另一个 DB2 数据库中。这些 PC/IXF 文件既可以被导入或装载到同种系统上的其他本地 DB2 数据库中；也可以被传递到其他操作系统平台上，并导入或装载到这种平台上的 DB2 数据库中；DB2MOVE 工具在导出操作中所生成的文件可以用作后来这些导入或装载操作的输入文件(参见表 6-7)。要使 DB2MOVE 操作成功执行，所使用的用户 ID 必须具有底层 DB2 数据移动工具所需要的适当授权。在调用 DB2MOVE 命令之前，并不需要数据库连接；该工具会为您建立数据库连接。

这个命令中支持的动作有 EXPORT、IMPORT、LOAD 和 COPY。DB2MOVE 的语法很简单。

db2move 命令的基本语法如下所示：

```
db2move <database-name> <action> [<option> <value>]
```

首先，您必须指定数据库名(想要移动的表所在的数据库)和要执行的操作(EXPORT、IMPORT、LOAD 和 COPY)。然后指定一个选项来定义操作的范围。例如，可以将一个操作限制在特定的表(-tn)、表空间(-ts)、表创建者(-tc)或模式名(-sn)范围内。指定表、表空间或表的创建者的一个子集只对 EXPORT 操作有效。如果指定多个值，就必须使用逗号将它们分隔开；值列表项之间不允许有空格。可以指定的项最多为 10 个。另外，也可以指定-tf 选项，此时要使用一个文件名作为参数，其中列出了要导出的表名；在该文件中，每行只能列出一个完整的表名。您还可以指定以下内容：

```
-io import-option
```

指定 DB2 的 IMPORT 工具可以运行的一种模式。有效的选项有：CREATE、INSERT、INSERT_UPDATE、REPLACE 和 REPLACE_CREATE。默认值为 REPLACE_CREATE。

```
-lo load-option
```

指定 DB2 的 LOAD 工具可以运行的一种模式。有效的选项有：INSERT 和 REPLACE。默认值为 INSERT。

```
-l lobpaths
```

指定要创建或查找的 LOB 文件的位置。必须指定一个或多个绝对路径名。如果指定了多个绝对路径，就必须使用逗号将它们分隔开；值之间不允许有空格。默认值是当前目录。

```
-u userid
```

指定一个用户 ID，DB2MOVE 工具可以使用这个用户 ID 登录到远程系统上。

```
-p password
```

指定对该用户进行认证的密码；DB2MOVE 工具需要使用一个有效的用户 ID 和密码登录到远程系统上。

下面是一些例子。DB2MOVE 命令用指定的用户 ID 和密码以 REPLACE 模式导入 sample 数据库中的所有表：

```
db2move sample IMPORT -io REPLACE -u userid -p password
```

下面的命令以 REPLACE 模式装载 db2admin 和 db2inst1 这两个模式下的所有表：

```
db2move sample LOAD -sn db2admin,db2inst1 -lo REPLACE
```

DB2MOVE 在执行期间会在所在目录生成一些文件，这些文件如表 6-8 所示。

表 6-8　DB2MOVE 在 EXPORT、IMPORT 和 LOAD 操作过程中需要或生成的文件

EXPORT	IMPORT		LOAD	
无输入文件，只有输出文件	输入文件	输出文件	输入文件	输出文件
EXPORT.out ① db2move.lst ② tab n.ixf ③ tab n.msg ④ tab na.nnn ⑤ system.msg ⑥		IMPORT.out ①		LOAD.out ①
	db2move.lst ②		db2move.lst ①	
	tab n.ixf ③		tab n.ixf ③	
		tab n.msg ④		tab n.msg ④
	tab na.nnn ⑤		tab na.nnn ⑤	

注：

① 包含一个对全部操作的摘要(ASCII 格式)。

② 包含一个源数据库中源表名的列表、对应的 PC/IXF 文件名和消息文件名(ASCII 格式)。

③ 包含从一个用户表中导出的数据，使用 n 标识(二进制格式)。

④ 包含对用户表请求操作的消息，使用 n 标识(ASCII 格式)。

⑤ 包含对用户表导出的大对象(LOB)数据，使用 n 标识。该文件的扩展名是一个数字，范围从 001 到 999；其中 a 是一个字符。这些 LOB 文件只有在被导出的表中包含 LOB 数据时才会被创建，保存在 LOB 路径目录中(二进制格式)。

⑥ 包含系统消息，只有在执行 EXPORT 操作并且已经指定好 LOB 路径时才会被创建(ASCII 格式)。

6.6.2　DB2 DDL 提取工具(DB2LOOK)

DB2LOOK 工具提取了 DDL 语句，可以用于在其他系统上重建数据库对象。在调用 DB2LOOK 命令之前，不需要提前建立数据库连接；这个工具会为您建立数据库连接。

db2look 命令的基本语法如下所示：

```
db2look -d <database-name> [<option1> <option2 <option>]
```

首先，您必须指定想要描述的对象所在的数据库名。然后指定一个或多个选项(可以是任意顺序)来定义提取的范围，包括：

- **-e** 提取数据库对象的 DDL 语句，例如表、视图、自动摘要表、索引、触发器、序列、主键、引用、检查约束、用户定义函数和过程。

- **-a** 提取用户创建的所有对象的 DDL 语句。如果这个选项与 **-e** 选项一起指定，那么就要对数据库中的所有对象都进行处理。
- **-z schema-name** 将输出限制为具有指定模式名的对象。
- **-t table-name** 将输出限制在一个或多个(最多 30 个)指定的表中。表名必须使用空格字符分隔开。
- **-m** 生成需要的 UPDATE 语句，对表、列和索引的统计信息进行复制。
- **-l** 为用户定义的表空间、数据库分区组和缓冲池生成 DDL 语句。
- **-x** 生成对数据库对象进行授权或回收权限的 DDL 语句。
- **-td delimiter** 指定 DB2LOOK 工具使用的分隔符；默认为分号(;)。
- **-o file-name** 将输出结果写入一个文件。如果没有指定该选项，就将输出结果写入标准输出设备。
- **-i userid** 指定用户 ID，DB2LOOK 工具需要使用它登录到远程系统上。
- **-w password** 指定对该用户进行认证的密码；DB2LOOK 工具需要使用一个有效的用户 ID 和密码登录到远程系统上。

6.6.3 利用 DB2MOVE 和 DB2LOOK 移动数据案例

现在让我们来看一个实际的例子。在一个正在运行 DB2 V8.2.4 数据库的 AIX V5.3 的系统上有一个 PROD 数据库。PROD 有 5 个表：MOVIE、ACTOR、APPEARS_IN、DIRECTOR 和 DIRECTS，每个表中都包含数据。其中有些表上定义了主键，ACTOR 表上定义了一个检查约束：ACTOR_AGE 约束要求 ACT_YR_OF_BIRTH 列的值都必须在 2004 年之前。

注意：

IXF 格式虽然包含表和索引的定义，但是不包含检查约束的定义。

我们要将 PROD 数据库复制到一个同样也在运行 DB2 V8.2.4 的 Windows XP 系统上。我们采取的策略是首先使用 DB2MOVE 将所有表的数据导出为 PC/IXF 文件，然后使用 DB2LOOK 为现有的数据库对象捕获 DDL 语句，包括 ACTOR_AGE 检查约束，它不会包含在 PC/IXF 文件中。接下来使用 FTP 将从这些工具中得到的输出文件传递到 Windows 系统上，在 Windows 系统上重新创建数据库以及其中的对象，最后运行 DB2MOVE 工具来装载 PC/IXF 文件中所包含的数据。步骤如下：

(1) 在 AIX 上，运行 DB2MOVE，导出 PROD 数据库中所有用户表中的数据：

```
$/home/prodinst>db2move PROD export
*****  DB2MOVE  *****
Action:     EXPORT
```

```
Connecting to database PROD ... successful!  Server: DB2 Common Server V8.2.4
Binding package automatically ...
Bind file: /home/prodinst/sqllib/bnd/db2move.bnd
Bind was successful!
EXPORT:     44 rows from table "PRODINST  "."DIRECTS"
EXPORT:     76 rows from table "PRODINST  "."APPEARS IN"
EXPORT:     40 rows from table "PRODINST  "."MOVIE"
EXPORT:     70 rows from table "PRODINST  "."ACTOR"
EXPORT:     42 rows from table "PRODINST  "."DIRECTOR"
Disconnecting from database ... successful!
End time:  Fri Mar 12 23:04:18 2008
```

(2) 在 AIX 上，运行 DB2LOOK，为 PROD 数据库中的所有对象捕获 DDL 语句，并将输出结果写入一个名为 db2look.sql 的文件中：

```
$/home/prodinst>db2look -d PROD -e -a -o db2look.sql
-- Generate statistics for all creators
-- Creating DDL for table(s)
-- Output is sent to file: db2look.sql
-- Binding package automatically ...
-- Bind is successful
```

(3) 在 Windows 上，使用 FTP 登录到 AIX 系统上，并下载最后的 DB2MOVE 操作所需要的输入文件。请确保使用二进制模式传输 PC/IXF 文件，使用 ASCII 模式传输 db2move.lst 文件和 db2look.sql 文件：

```
ftp> prompt
Interactive mode Off .
ftp> bin
200 Type set to I.
ftp> mget *.ixf
200 Type set to I.
200 PORT command successful.
150 Opening data connection for tab1.ixf (4513 bytes).
226 Transfer complete.
ftp: 4513 bytes received in 0.13Seconds 34.45Kbytes/sec.
 ...
200 PORT command successful.
150 Opening data connection for tab5.ixf (6289 bytes).
226 Transfer complete.
ftp: 6289 bytes received in 0.12Seconds 52.41Kbytes/sec.
ftp> asc
200 Type set to A; form set to N.
```

```
ftp> get db2move.lst
200 PORT command successful.
150 Opening data connection for db2move.lst (205 bytes).
226 Transfer complete.
ftp: 210 bytes received in 0.01Seconds 21.00Kbytes/sec.
ftp> get db2look.sql
200 PORT command successful.
150 Opening data connection for db2look.sql (2754 bytes).
226 Transfer complete.
ftp: 2876 bytes received in 0.15Seconds 19.17Kbytes/sec.
ftp> bye
221 Goodbye.
```

(4) 在 Windows XP 上，创建 PROD 数据库，然后运行 DB2LOOK 工具所生成的脚本创建数据库对象，包括用户表和检查约束 ACTOR_AGE。运行 DB2MOVE，将数据从 PC/IXF 文件装载到 PROD 数据库中的所有用户表中：

```
C:\Db2move>db2 create db PROD
C:\Db2move>db2 -tvf db2look.sql
C:\Db2move>db2move PROD load
*****  DB2MOVE  *****
Action:     LOAD
Connecting to database PROD ... successful!  Server: DB2 Common Server V8.2.4
Binding package automatically ...
Bind file: C:\Program files\IBM\SQLLIB\BND\DB2MOVE.BND
Bind was successful!
* LOAD:  table "PRODINST  "."DIRECTS"
  -Rows read:  44  -Loaded:  44  -Rejected   0  -Deleted   0  -Committed  44
* LOAD:  table "PRODINST  "."APPEARS_IN"
  -Rows read:  76  -Loaded:  76  -Rejected   0  -Deleted   0  -Committed  76
* LOAD:  table "PRODINST  "."MOVIE"
  -Rows read:  40  -Loaded:  40  -Rejected   0  -Deleted   0  -Committed  40
* LOAD:  table "PRODINST  "."ACTOR"
  -Rows read:  70  -Loaded:  70  -Rejected   0  -Deleted   0  -Committed  70
* LOAD:  table "PRODINST  "."DIRECTOR"
  -Rows read:  42  -Loaded:  42  -Rejected   0  -Deleted   0  -Committed  42
Disconnecting from database ... successful!
End time:  Sat Mar 13 21:01:25 2008
```

(5) 在 Windows XP 上，验证所复制的 PROD 数据库是否完好无损，ACTOR_AGE 检查约束是否可以正常工作：

```
C:\Db2move>db2 connect to PROD
C:\Db2move>db2 select * from movie fetch first 5 rows only
MOVIE_ID TITLE                           YR_RELEASED
--------- ------------------------------ -----------
23154    Carousel                             1956
44524    El Cid                               1961
78456    Giant                                1956
45692    African Queen                        1951
67845    Casablanca                           1942
  5 record(s) selected.
C:\Db2move>db2 select * from actor fetch first 5 rows only
ACTOR ID ACTOR NAME          ACT YR OF BIRTH
-------- -------------------- ---------------
SQL0668N Operation not allowed for reason code "1" on table "PRODINST.ACTOR".
SQLSTATE=57016
C:\Db2move>db2 set integrity for actor immediate checked
C:\Db2move>db2 insert into actor values ('58825','Naomi Watts',2009)
DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:
SQL0545N  The requested operation is not allowed because a row does not
satisfy the check constraint "PRODINST.ACTOR.ACTOR_AGE".  SQLSTATE=23513
C:\Db2move>db2 set integrity for prodinst.actor immediate checked
C:\Db2move>db2 insert into actor values ('58825','Naomi Watts',1968)
C:\Db2move>db2 "select * from actor where act yr of birth > 1960"
ACTOR_ID ACTOR_NAME          ACT_YR_OF_BIRTH
-------- -------------------- ---------------
58825    Naomi Watts                     1968
46739    Elaine Cassidy                  1980
44333    Adam Baldwin                    1962
44445    Tom Cruise                      1962
  4 record(s) selected.
```

ACTOR 表最初处于一种检查挂起状态(由于检查约束的原因)，需要执行 db2 set integrity for prodinst.actor immediate checked 语句将其转换成正常状态。然而，后续的插入操作会失败，因为年份值违反了检查约束。最后我们使用一个有效的年份值，就可以成功插入了。

6.6.4 带 COPY 操作的 DB2MOVE 实用程序

DB2MOVE 命令中支持的动作有 EXPORT、IMPORT、LOAD 和 COPY。EXPORT、IMPORT 和 LOAD 这几个动作的行为与前面描述的完全相同。您可能不熟悉的唯一一个动作就是 COPY。它将一个或多个模式中的表复制到一个目标数据库中。在 COPY 动作中，可以用-sn 选项指定一个或多个模式。只有具有-sn 选项中指定的模式名的表才被复制(通过

导出)。带 COPY 操作的 DB2MOVE 实用程序的语法如下：

```
>>-db2move--dbname—COPY----+----------------------+-+-----><
                           +- -sn--schema-names-----+
                           +- -tn--table-names------+
                           +- -tf--filename---------+
                           +- -co--copy-option------+
                           +- -u--userid------------+
                           +- -p--password----------+
```

在使用带 COPY 操作的 DB2MOVE 实用程序时，可以在-co 后面使用以下选项：

- TARGET_DB <db name> [USER <userid> USING <password>]

允许用户指定目标数据库的名称以及用户名和密码(可以使用-p 和-u 选项指定源数据库的用户名和密码)。USER 或 USING 子句是可选的。如果 USER 指定一个用户 ID，那么必须在 USING 子句中提供密码；如果没有指定密码，那么 DB2MOVE 会提示输入密码。出现这个提示是由于后面要讨论的安全因素。TARGET_DB 是 COPY 操作的必要选项。TARGET_DB 不能与源数据库相同。COPY 操作要求至少输入一个模式(-sn)或一个表(-tn 或-tf)。

如果指定多个模式名，则使用逗号将它们隔开，这里不允许使用空格。请参考下面的例子：

```
db2move sample COPY -sn db2inst1，prodschema -co TARGET DB acctdb  USER nxz
USING nxz DDL_AND_LOAD
```

上面的 DB2MOVE 命令复制 db2inst1 和 prodschema 模式下受支持的对象。后面跟着的-co 选项使这个命令更加有趣。TARGET_DB 选项指定这些模式将被复制到的目标数据库。当指定了 COPY 动作时，这个选项是强制性的。此外，目标数据库必须不同于源数据库。当连接到目标数据库时，可以通过 USER 和 USING 选项提供用户名和密码。

- MODE

DDL_AND_LOAD：从源模式创建支持的所有对象，并用源表数据填充表。这是默认选项。

DDL_ONLY：从源模式创建支持的所有对象，但是不重新填充表。

LOAD_ONLY：将指定的所有表从源数据库装载到目标数据库中。这些表必须已经在目标数据库中存在。

- SCHEMA_MAP

允许用户在向目标数据库进行复制时对模式重新命名。需要提供一个源-目标模式映射列表，映射由括号包围，由逗号分隔。例如，schema_map ((s1，t1)，(s2，t2))。这意味着

把模式 s1 中的对象复制到目标数据库中的模式 t1 中，把模式 s2 中的对象复制到目标数据库中的模式 t2 中。默认的目标模式名与源模式名相同，这也是推荐的做法。这是因为 DM2MOVE 不会尝试修改对象体中的任何限定对象的模式。因此，如果对象体中有限定对象，那么不同的目标模式名就可能导致问题。

在使用 SCHEMA_MAP 选项时要特别小心。只有对象本身的模式被重命名，而对象体中的对象仍保持不变。例如：

```
CREATE VIEW FOO.v1 AS 'SELECT c1 FROM FOO.T1'
```

将模式从 FOO 重命名为 BAR 将导致：

```
CREATE VIEW BAR.v1 AS 'SELECT c1 FROM FOO.T1'
```

如果 FOO.T1 没有定义，那么目标数据库中就不能成功地创建 BAR.v1。

- NONRECOVERABLE

这个选项允许用户改变 COPY-NO 装载操作的默认行为。在采用默认行为时，用户必须对进行装载的每个表空间进行备份。如果指定这个 NONRECOVERABLE 关键字，那么用户不必马上对表空间进行备份。但是，强烈建议尽快进行备份，从而确保新创建的表可以被正确地恢复。

- OWNER

允许用户在成功地 COPY 操作之后修改在目标模式中创建的每个新对象的所有者。目标对象默认的所有者是进行连接的用户。如果指定这个选项，那么所有者就改为新的所有者。

- TABLESPACE_MAP

用户可以指定在复制期间使用的表空间名映射，而不使用来自源数据库的表空间。这里应该提供一个表空间映射数组，映射由括号包围。例如，tablespace_map((TS1，TS2)，(TS3，TS4))。这意味着把表空间 TS1 中的所有对象复制到目标数据库中的表空间 TS2 中，把表空间 TS3 中的对象复制到目标数据库中的表空间 TS4 中。如果是((T1，T2)，(T2，T3))，那么源数据库中 T1 中的所有对象在目标数据库中的 T2 中重新创建，源数据库中 T2 中的所有对象在目标数据库中的 T3 中重新创建。默认情况下，使用与源数据库相同的表空间名，在这种情况下，不必提供这个表空间的映射。如果指定的表空间不存在，那么使用这个表空间的对象复制操作会失败，这一情况会记录在错误文件中。

用户还可以使用 SYS_ANY 关键字，这表示应该使用默认的表空间选择算法来选择表空间。在这种情况下，DB2MOVE 可以选择任何可用的表空间作为目标表空间。SYS_ANY 关键字可以用于所有表空间。例如，tablespace_map SYS_ANY。另外，用户可以为某些表

空间指定特定的映射，而对其他表空间使用默认的表空间选择算法。例如，tablespace_map ((TS1，TS2)，(TS3，TS4)，SYS_ANY)。这表示表空间 TS1 映射为 TS2，TS3 映射为 TS4，而其他表空间将使用默认的表空间目标。使用 SYS_ANY 关键字是因为表空间名称不可能以“SYS”开头。

让我们来看一个综合性的例子。

```
    db2move sample COPY -sn db2inst1,prodschema    -co TARGET DB acctdb USER nxz
USING nxzpwasswd LOAD_ONLY    SCHEMA_MAP
((db2inst1,db2inst2),(prodschema,devschema))
        TABLESPACE_MAP SYS_ANY    NONRECOVERABLE
```

这个命令将 db2inst1 和 prodschema 中受支持的对象从 SAMPLE 数据库复制到 ACCTDB 数据库。用户名 nxz 和密码 nxzpasswd 用于连接到 ACCTDB。目标表已经存在于 ACCTDB 中，这些表将被重新填充。db2inst1 和 prodschema 模式下的所有对象现在分别在 db2inst2 和 devschema 模式下。最后不使用 SAMPLE 数据库中定义的表空间名称，而是使用 ACCTDB 中默认的表空间。

NONRECOVERABLE 选项允许用户在复制完成之后立即使用装载的目标表空间。这里不要求备份表空间，但是强烈建议在早期方便的时候做一个备份。

使用 DB2MOVE COPY 选项案例

在下面这个示例中，PROD 模式中的数据库修改成功地通过了测试，下面要把新的提交版本复制到一个新数据库并命名为 DEV 模式。

使用示例：

```
    db2move BANKDEV COPY -sn PROD -co TARGET  DB BANKSHIP USER   nxz USING nxzpasswd
MODE DDL_ONLY SCHEMA_MAP ((PROD,DEV))
```

只将模式 PROD 中的对象从源数据库 BANKDEV 复制到目标数据库 BANKSHIP 中的模式 DEV 中：

```
    db2move BANKDEV COPY -sn PROD -co TARGET_DB BANKSHIP USER   nxz USING nxzpasswd
SCHEMA_MAP ((PROD,DEV))
```

将模式 PROD 中的对象和包含的数据从源数据库 BANKDEV 复制到目标数据库 BANKSHIP 中的模式 DEV 中：

```
    db2move BANKDEV COPY -sn PROD -co TARGET DB BANKSHIP USER USER nxz USING
nxzpasswd   SCHEMA_MAP ((PROD,DEV)) OWNER V9_ADMIN   TABLESPACE_MAP
((USERSPACE1,V9_USERSPACE1),(USERSPACE2,V9_USERSPACE2),SYS_ANY)
```

将模式 PROD 中的对象和包含的数据复制到模式 DE 中，并指定新的对象所有者 V9_ADMIN。将表空间 USERSPACE1 中的对象复制到表空间 V9_USERSPACE1 中。将表空间 USERSPACE2 中的对象复制到表空间 V9_USERSPACE2 中。其他表空间使用默认的表空间选择算法。

带 COPY 操作的 DB2MOVE 实用程序生成的文件：

- COPYSCHEMA.timestamp.msg：DB2MOVE COPY 操作产生的消息。
- COPYSCHEMA.timestamp.err：DB2MOVE COPY 操作产生的错误(只在发生错误的情况下生成这个文件)。
- LOADTABLE.timestamp.msg：DB2MOVE COPY 操作中 LOAD 操作产生的消息(MODE DDL_AND_LOAD 和 LOAD_ONLY)。
- LOADTABLE.timestamp.err：DB2MOVE COPY 操作中 LOAD 操作产生的错误(只在发生错误的情况下生成这个文件)。

例 6-16 成功执行带 COPY 操作的 DB2MOVE 实用程序所产生的消息：

```
Application code page not determined, using ANSI codepage 1386
*****  DB2MOVE  *****
Action:  COPY
Start time:  Mon Jun 12 17:46:39 2008
All schema names matching:  PROD;
Connecting to database BANKDEV ... successful!  Server : DB2 Common Server V9.5.0
Copy schema PROD to DEV on the target database BANKSHIP
Create DMT :  "SYSTOOLS"."DMT 448d83d5c76c"
db2move finished successfully
Files generated:
-----------------
COPYSCHEMA.20080612174639.msg
LOADTABLE.20080612174639.MSG
Please delete these files when they are no longer needed.
End time:  Mon Jun 12 17:46:43 2008
```

例 6-17 不成功地执行带 COPY 操作的 DB2MOVE 实用程序所产生的消息：

```
Application code page not determined, using ANSI codepage 1386
*****  DB2MOVE  *****
Action:  COPY
Start time:  Mon Jun 12 17:04:57 2008
All schema names matching:  PROD;
Connecting to database BANKDEV ... successful!  Server : DB2 Common Server V9.0.0
Copy schema PROD to DEV on the target database BANKSHIP
Create DMT :  "SYSTOOLS"."DMT_448d829c49bdd"
```

```
Rolled back all changes from the create phase (debuginfo:140).
db2move failed with -1 (debuginfo:220).
Files generated:
-----------------
COPYSCHEMA.20080612170457.msg
COPYSCHEMA.20080612170457.ERR
Please delete these files when they are no longer needed.
**Error occurred -1
End time:  Mon Jun 12 17:05:01 2008
Content of file COPYSCHEMA.20080612170457.ERR:
1 Schema        : PROD  .TEST
  Type          : TABLE
  Error Msg     : [IBM][CLI Driver][DB2/NT] SQL0204N
  "USERSPACE2" is an undefined name. SQLSTATE=42704
  DDL           :
 CREATE TABLE "DEV  "."TEST" (  "COL1" INTEGER )    IN "USERSPACE2"
```

DB2MOVE 使用说明和限制

- DB2MOVE 实用程序尝试复制所有允许的模式对象，但是以下类型的对象除外：
  - ◊ 表层次结构
  - ◊ Java 例程存档(JARS)
  - ◊ 昵称(nickname)
  - ◊ 应用程序包
  - ◊ 视图层次结构
  - ◊ 对象特权(创建的所有新对象都具有默认的授权)
  - ◊ 统计信息(新对象不包含统计信息)
  - ◊ 索引扩展(与用户定义的结构化类型相关)
  - ◊ 用户定义的结构化类型及其转换函数
- 在复制处理期间修改源模式中的表可能会导致在复制操作之后目标模式中的数据不相同。
- 对于不与模式相关的对象来说(比如表空间和事件监视器)，在模式复制操作期间不进行处理。
- 在对复制的(replicated)表进行复制时，表的新副本没有启用订阅。表只作为常规表重新创建。
- 如果源数据库和目标数据库不在同一个实例中，那么必须对源数据库进行编目。

- 运行多个 DB2MOVE 命令将模式从一个数据库复制到另一个数据库会导致死锁。每次应该只执行一个 DB2MOVE 命令。

6.7　本章小结

本章我们讲解了 IMPORT、EXPORT、LOAD 和 DB2MOVE 几种数据移动工具。其实 IMPORT 和 EXPORT 非常简单。LOAD 相对来说比较复杂，所以本章我们用了很大的篇幅来讲解 LOAD。在本章的最后，我们给大家讲解了 DB2MOVE 及其应用案例。大家应灵活掌握这些工具以便根据自己的需要选用最合适的工具来移动数据。



数据库备份与恢复

在以 IT 架构为基础的现代企业环境中，数据是企业最宝贵的资源。数据的丢失往往意味着难以计量和弥补的损失。从这一角度出发，对数据进行备份与恢复，是数据库管理系统最重要的功能之一。是否具有灵活、健全的数据备份与恢复机制也是当前衡量一个关系数据库系统是否稳定，是否安全的重要指标。

为了更好地保护客户的关键数据，DB2 数据库提供了健全的备份、恢复功能，以方便用户制订并实施满足自己业务需求的数据保护策略。

本章主要讲解如下内容：

- 恢复概念
- DB2 日志
- 数据库和表空间备份
- 数据库和表空间恢复
- 数据库和表空间前滚
- RECOVER 数据库实用程序
- 数据库重建
- 监控备份、恢复和复原
- 优化备份、恢复和复原性能

7.1 恢复概念

备份恢复概念

数据库的备份是数据库的副本以及一些控制信息，在出现故障的情况下，可以随时用它进行恢复。数据库备份最小化了数据丢失，能够让您使用恢复过程，从备份副本中重新

构造失败的数据库。有多种类型的错误导致需要恢复数据库。但不是所有类型的错误需要人工干预。下面是您可能会碰到的各种类型的故障恢复场景：

- 语句失败

当应用程序中存在逻辑错误时，就会出现语句错误。出现这种失败可能有许多不同的原因，例如，语句无限地循环运行，用户不具备执行某些任务所需的正确特权(导致有效的语句失败)，或者由于空间不足而导致的插入失败等。

- 用户错误

对于数据库应用程序，有效但具破坏性的语句(比如删除整个工资单表中的记录，误删除重要表，或者意外删除整个数据库)可能导致长时间停机。要避免这些错误，需要更多的正确培训和指导。数据定义语言(DDL)语句是不能回滚的语句。因此，如果用户的错误涉及 DDL，那么将需要采取正确的措施来恢复数据库。

例如，如果错误删除(drop)了表，那么作为 DBA，您有两种选择。您可以使用备份时刻的副本，并将它前滚至删除表之前的某个时间点(时间点恢复)；或者您也可以通过 EXPORT 实用程序恢复逻辑备份。这两种选择都可能潜在地导致数据丢失。当用户删除 DB2 中的表时，您可以执行数据库级时间点恢复，前滚至删除表之前的时间点；或者最好使用表空间级的前滚操作，这样，您就不必让数据库停止服务，您的用户仍可以访问其他表空间中的数据。

- 进程失败

用户、服务器或后台进程的失败是由于这些进程的异常终止，或者是因为从这些进程中断开连接。失败的进程将导致无法继续完成任务。在 DB2 上，像 db2agent 失效这样的进程失败可能会导致崩溃恢复。在崩溃恢复中，将把已提交的数据更新到磁盘上，并回滚没有提交的事务。

- 数据库实例崩溃

数据库实例失败通常是由操作系统崩溃或停电引起的。在 DB2 中，当数据库管理器和内存结构由于电源故障、磁盘损坏或网络故障而不能正常工作时，将需要通过崩溃恢复将 DB2 恢复到一致可用的状态。

- 存储介质故障

当有人无意从文件系统删除了数据库文件时，就会发生媒介失败，这时，整个硬盘及其数据文件都无法正常工作，从而导致无法访问数据，有时系统还会发生纯粹的数据损坏。相对于我们已经讨论的类型，这是更加严重的一类失败。

- 灾难

放置系统的设施遭到火灾、洪水、地震或其他类似灾难的毁坏。

在 DB2 中，这些失败的处理方式大致是一样的，都是通过脱机或联机备份并使用前滚

操作恢复日志来完成的。您永远也无法知道您的系统何时会碰到灾难或故障。因此最好早作准备，不但要防止数据受到外部因素的影响，也要防止内部用户有意或无意中用不正确的信息破坏您的数据库。请考虑下列问题：您有备份您的数据库吗？您能够恢复执行到最后一秒的所有事务吗？

为了尽量减少丢失的数据，需要有一个恢复策略，确保这个恢复策略可行，并不断地加以演练。

恢复策略

为了制订恢复策略，应该问一问自己下面这些问题：

- 您的数据可以从另一个地方装载吗？
- 您能承受多少数据的丢失？
- 您需要花多少时间才能恢复数据库？
- 您有什么可用的资源来存储备份和日志文件？

事务

在讲解恢复类型之前，我们首先要了解事务的概念，事务、交易和工作单元(UOW)其实都是一个概念。关系数据库为了保证数据库的可恢复性和一致性引入了事务这个概念。事务具有原子性、一致性、隔离性和永久性这几个特性。

- 原子性：事务的原子性指的是，事务中包含的 SQL 操作作为数据库的逻辑工作单元(UOW)，它所做的对数据的修改操作要么全部成功，要么全部失败。也就是说事务的操纵序列或者完全应用到数据库或者完全不影响数据库。这种特性称为原子性。
- 一致性：事务的一致性指的是，在一个事务执行之前和执行之后数据库都必须处于一致状态。一致性处理数据库中对所有语义约束的保护。假如数据库的状态满足所有的完整性约束，就说该数据库是一致的。例如，当数据库处于一致性状态 S1 时，对数据库执行一个事务，在事务执行期间假定数据库的状态是不一致的，当事务执行结束时，数据库处在一致性状态 S2。
- 隔离性：隔离性指并发的事务是相互隔离的。即一个事务内部的操作及其正在操作的数据必须封锁起来，不被其企图进行修改的事务看到。隔离性是 DBMS 针对并发事务间的冲突提供的安全保证。DBMS 可以通过加锁在并发执行的事务间提供不同级别的隔离。如果对并发交叉执行的事务没有任何控制，那么操纵相同的共享对象的多个并发事务的执行可能会引起异常情况。这个概念主要在锁和并发

中用到，在本章备份恢复部分用不到这个概念，我们会在“第 9 章：锁和并发”部分讲解它。

- 永久性：永久性意味着一旦一个事务提交，DBMS 保证它对数据库中数据的改变应该是永久性的，即使数据库发生存储介质故障。永久性确保已提交事务的更新不会丢失，即对已提交事务的更新能恢复。

在 DB2 数据库中，事务是最小的交易单位和恢复单位。一个事务由一条或多条 SQL 语句组成，最后是一条 COMMIT 或 ROLLBACK 语句。事务中的所有语句被看做一个单元，事务中所包含的 SQL 语句要么全部成功，要么全部失败，以确保数据的原子性和一致性。例如，一个客户试图将 1000 元从一个储蓄账户转到一个支票账户，在这种情况下，事务是这样的：

```
DELETE 1000 yuan from SAVINGS  account
INSERT 1000 yuan  to  CHECKING account
COMMIT
```

如果这些语句没有被当作一个单元，那么可以想象一下：在 DELETE 之后、INSERT 语句之前系统突然停电，会出现什么情况？这个客户将丢失 1000 元。但是，如果将这些语句当作一个单元，就不会发生这样的事情。DB2 将知道这个单元没有完成(COMMIT)，因此它将回滚(rollback)之前的语句作出的所有更改，并将受影响的行返回到事务开始之前的状态。

COMMIT 或 ROLLBACK 用来显式地提交或回滚一个事务。就像上面我们举的那个例子，在这个事务中有两条 SQL 语句，这两条 SQL 语句要么全部成功、要么全部失败以保证交易的原子性。假设我们在执行完第一条 SQL 语句后突然机器停电了，那么数据库会自动执行 ROLLBACK，撤销(undo)前面已经执行的第一条 SQL 语句，这个工作由数据库自动来完成。如果这两条 SQL 语句执行后我们显式地执行了 COMMIT，那么对数据库的更改将永久生效，这就叫交易的永久性。假设我们提交后，更改的数据还没有来得及写到硬盘上，突然机器又停电了，这个没关系，因为一旦我们提交(COMMIT)了，我们对数据库所做的操作就已经记录到数据库日志中了，数据库下一次重新启动时，数据库会自动做 redo 操作。redo 就是从数据库日志中把刚才已经提交但是还没有来得及写到硬盘的数据更改重新再做一次。这个操作也是由数据库自动来完成的。当然，如果你的数据库日志也损坏了，那就没有办法执行 redo 了。

恢复的类型

了解完事务的概念后，接下来让我们看看针对我们上面所讲的几种故障场景，DB2 数据库中的几种恢复类型。

7.1.1　崩溃恢复

对数据库执行的事务(也称工作单元)可能被意外中断。如果在作为工作单元一部分的所有更改完成和提交之前发生故障，那么该数据库就会处于不一致和不可用的状态。崩溃恢复是将数据库恢复为一致并可用状态的进程。为此，回滚未提交的事务，并重做当发生崩溃时仍在内存中的已提交事务(图 7-1 所示)。当数据库处于一致和可用状态时，它处于一种被称为“一致点”的状态。

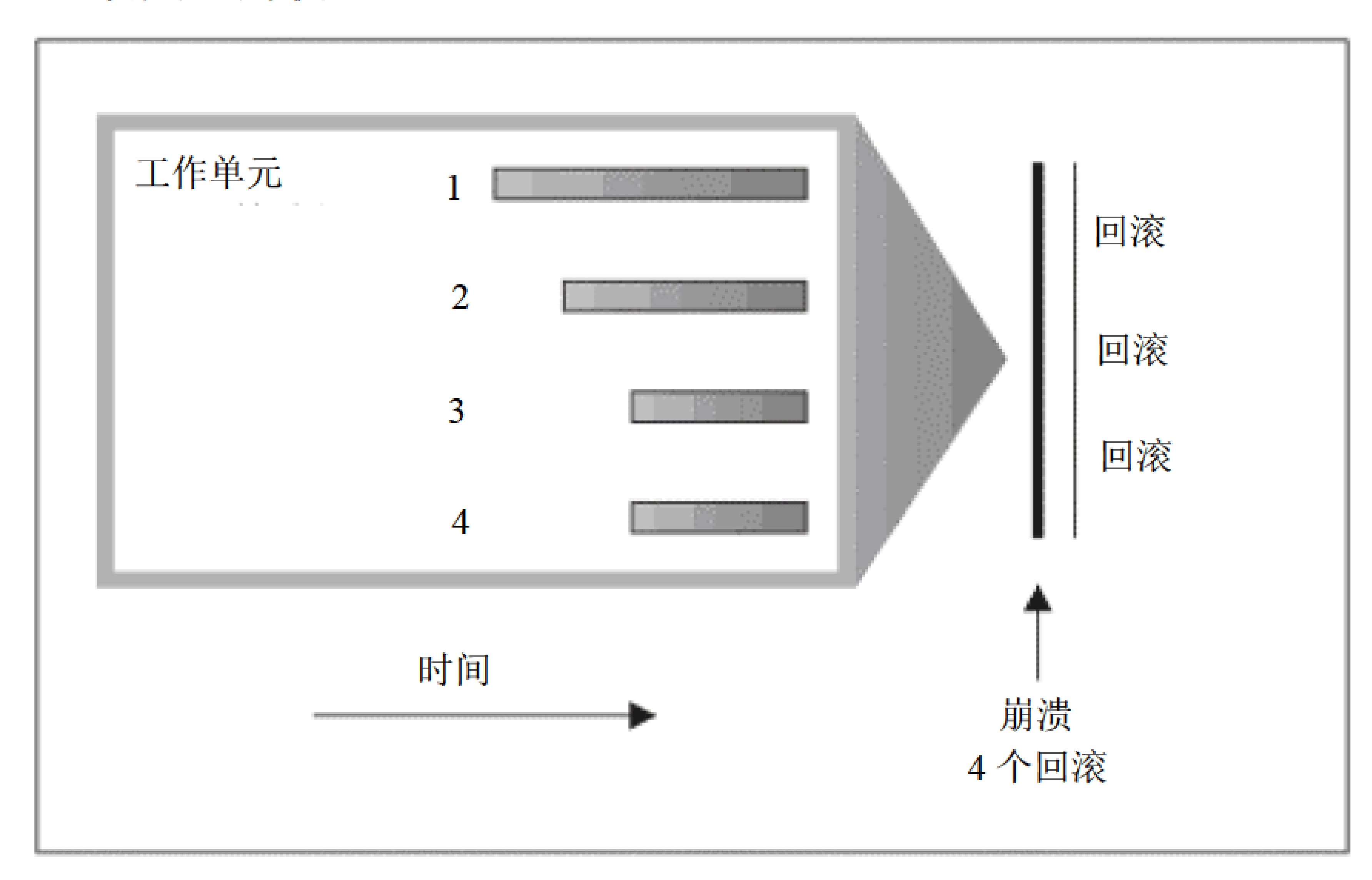


图 7-1　回滚工作单元(崩溃恢复)

事务处理失败是由于出现了严重错误或导致数据库或数据库管理器异常结束的情况。部分完成的工作单元或发生故障时未写到磁盘中的 UOW 使数据库处于不一致状态。在事务处理故障之后必须恢复数据库。导致事务处理故障的情况有：

- 机器上的断电故障，它会导致使用该机器的数据库管理器和数据库分区崩溃
- 硬件故障，例如内存毁坏，磁盘、CPU 或网络故障。
- 导致 DB2 崩溃的严重操作系统错误
- 应用程序异常终止

如果您希望崩溃恢复(不完整工作单元)的回滚是由数据库管理器自动完成的，那么应将 *autorestart* 数据库配置参数设置为 ON，以启用该自动重新启动参数(这是默认值)。如果不想要重新启动行为，那么将 *autorestart* 数据库配置参数设置为 OFF，但这样一来，您需要在数据库故障发生时发出 RESTART DATABASE 命令。如果数据库 I/O 在发生崩溃之前已处于暂挂状态，那么必须指定 RESTART DATABASE 命令的 WRITE RESUME 选项才能使崩溃恢复继续进行。

如果对用于前滚恢复(稍后讲解)的数据库应用崩溃恢复(即，未将 logarchmeth1 配置参数设置为 OFF)，且在崩溃恢复期间个别表空间发生错误，那么会让该表空间脱机，崩溃恢复继续进行。该表空间直到修复后才能对其进行访问。在崩溃恢复完成时，该数据库中的其他表空间将是可访问的，并且可以与该数据库建立连接。但是，如果脱机的表空间包含系统目录，那么必须先修复它才允许进行所有连接。

通过 undo(回滚)未提交的事务，使处于不一致状态的数据库恢复到一致状态。再次考虑我们前面举的例子。如果在 COMMIT 语句之前出现停电事故，则下一次 DB2 重新启动并访问数据库时，DB2 将首先回滚 INSERT 语句，然后回滚 DELETE 语句(回滚语句的顺序与这些语句当初执行的顺序相反)。

7.1.2 灾难恢复

术语灾难恢复用于描述在发生火灾、地震、恶意破坏或其他大灾害时复原数据库所需要执行的活动。灾难恢复计划可以包括以下其中一项或多项：

- 在紧急情况下使用的场所
- 用于恢复数据库的另一台机器
- 以非现场方式存储数据库备份和/或表空间备份以及归档日志

保护数据完整性或防止整个站点故障的一种方法是实现 DB2 高可用性灾难恢复(HADR)功能。设置此功能后，HADR 通过将数据更改从源数据库(称为主数据库)复制到目标数据库(称为备用数据库)来防止数据丢失。也可以使用存储器镜像功能(例如远程复制 PPRC 或 SRDF)来保护数据。PPRC 或 SRDF 提供了卷或磁盘同步复制功能来预防灾难。

注意：

灾难恢复对于重要系统来说至关重要。灾难恢复属于高可用的概念，这超出了本书的讲解范围。

7.1.3 版本恢复

当数据库发生存储故障时，我们可以采用版本恢复。版本恢复指的是使用备份操作期间创建的映像来复原数据库的先前版本。版本恢复允许我们使用从 BACKUP 命令获得的一个备份镜像恢复前一个版本的数据库。被恢复的数据库将包含执行 BACKUP 命令时该数据库所处状态的信息。备份之后执行的活动信息将丢失。对不可恢复数据库(即，该数据库没有归档日志)使用此方法。还可通过对 RESTORE DATABASE 命令使用 WITHOUT ROLLING FORWARD 选项，来对可恢复数据库使用此方法。数据库复原操作将使用先前创建的备份映像来复原整个数据库。数据库备份使您可以将数据库复原到与进行备份时完全相同的状态。但是，从备份时间到故障时间之间的每个工作单元都将丢失(请参阅图 7-2)。

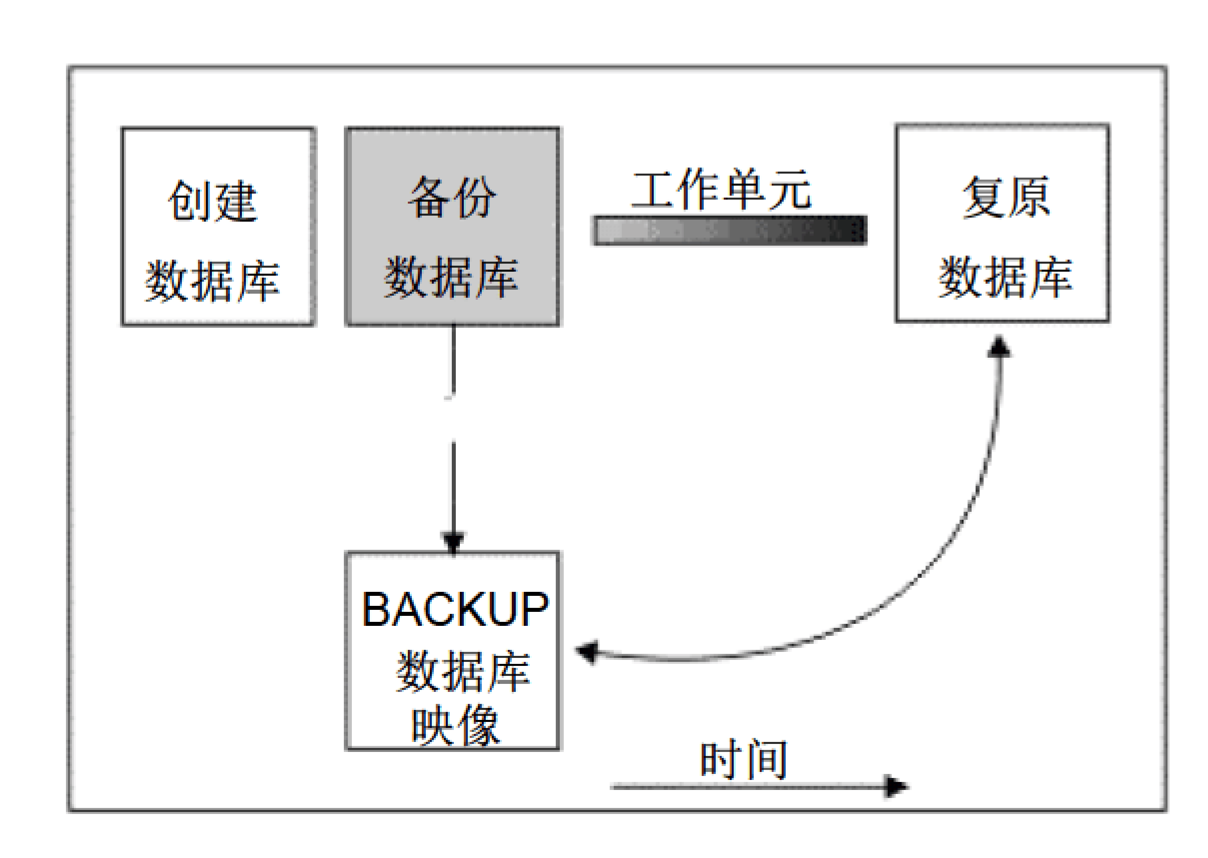


图 7-2　从进行备份开始到发生故障时为止的所有工作单元将丢失

使用版本恢复方法，必须定期安排和执行完整数据库备份。

7.1.4　前滚恢复

如果发生了存储故障或者用户不小心误删除了表中的重要数据。这种情况下需要用到前滚恢复。要使用前滚恢复方法，必须已建立数据库的备份，并且已归档日志(方法是将 *logarchmeth1* 和 *logarchmeth2* 配置参数设置为 OFF 之外的值)。复原数据库并指定 WITHOUT ROLLING FORWARD 选项等效于使用版本恢复方法。此数据库被复原到创建脱机备份映像时的状态。如果复原数据库，但没有对复原数据库操作指定 WITHOUT ROLLING FORWARD 选项，那么该数据库在复原操作结束时将处于前滚暂挂状态。这允许进行前滚恢复。

要考虑的两种前滚恢复类型是：

- 数据库前滚恢复。在此类型的前滚恢复中，记录在数据库日志中的事务应用到以下数据库复原操作中(请参阅图 7-3)。该数据库日志记录了对该数据库所作的所有更改。这种方法将数据库恢复到在某特定时间点的状态，或恢复到故障前的状态(即，恢复到活动日志的末尾)。

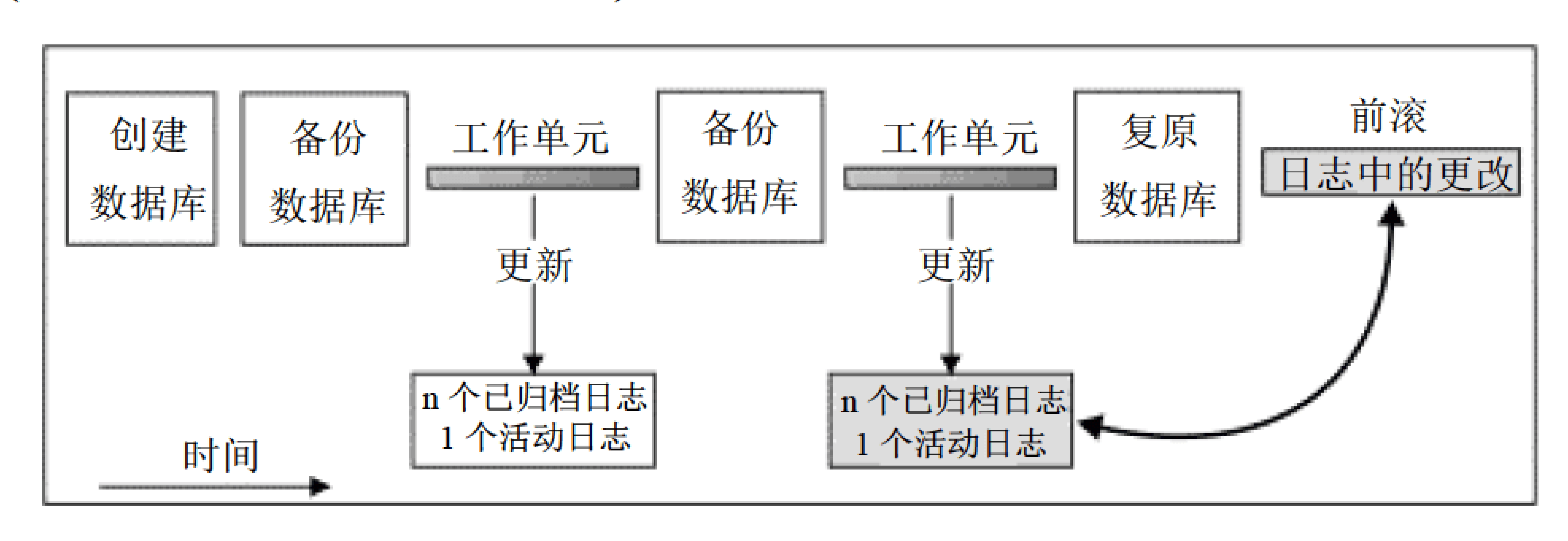


图 7-3　数据库前滚恢复(在运行时间较长的事务中，可以有多个活动日志)

● 表空间前滚恢复。如果启用了某个数据库，那么既可以对它进行前滚恢复，也可以对它进行备份、复原并前滚表空间(请参阅图 7-4)。要执行表空间复原和前滚操作，需要整个数据库(即所有表空间)或一个或多个个别表空间的备份映像。还需要影响要恢复的表空间的日志记录。可在日志中前滚至以下两点之一：
  ◊ 日志末尾
  ◊ 一个特定时间点(称为时间点恢复)

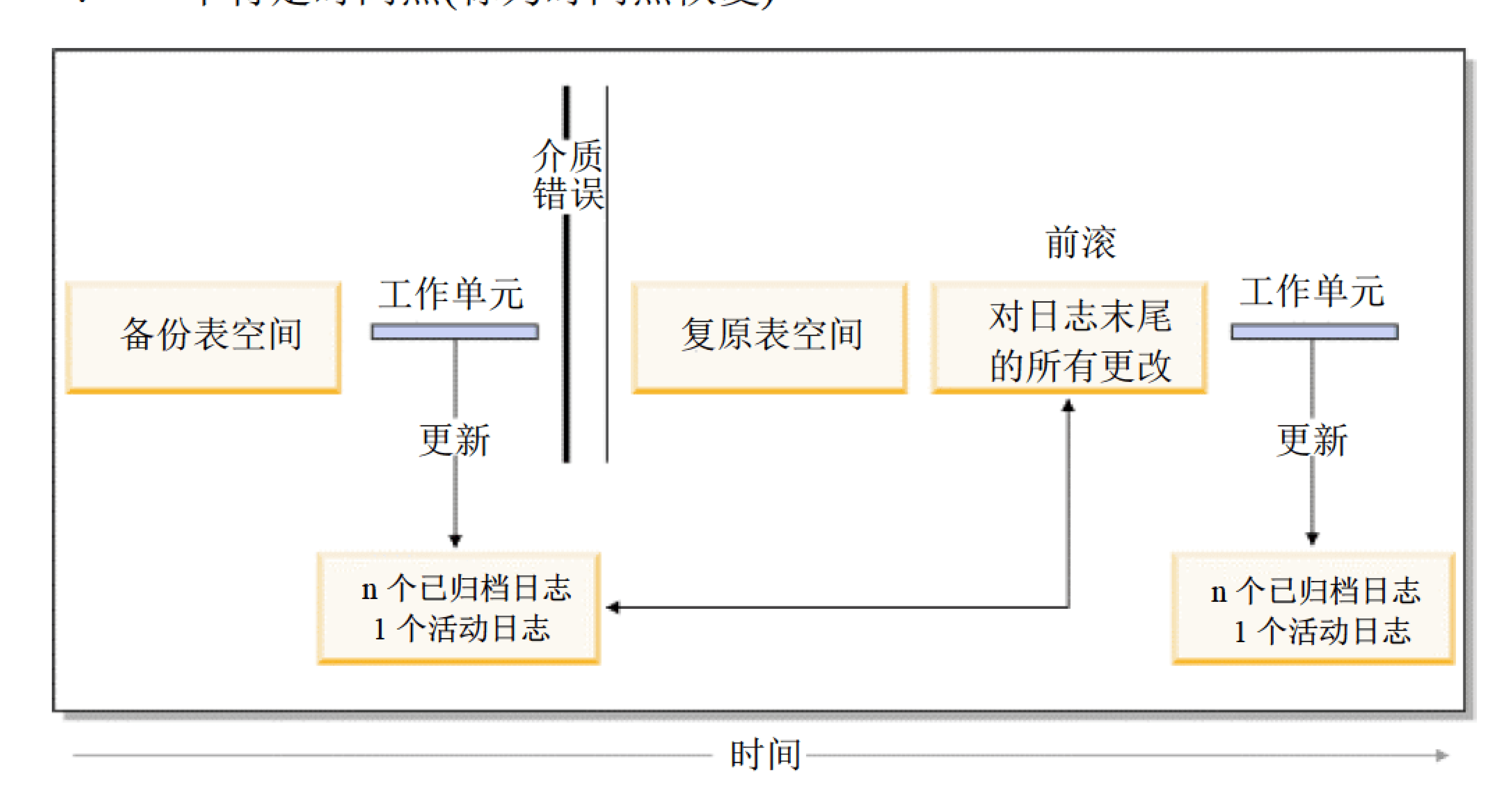


图 7-4　表空间前滚恢复(在运行时间较长的事务中，可以有多个活动日志)

可在下列两种情况下使用表空间前滚恢复：

● 在一个表空间复原操作后，该表空间始终处于前滚暂挂状态，且必须将它前滚。调用 ROLLFORWARD DATABASE 命令将日志应用于表空间以使其前滚至某个时间点或日志末尾。
● 如果一个或多个表空间在崩溃恢复后处于前滚暂挂状态，首先应校正该表空间的问题。某些情况下，校正表空间的问题不涉及复原数据库操作。例如，掉电可能导致表空间处于前滚暂挂状态。在这种情况下，复原数据库操作并不是必需的。一旦校正了该表空间的问题，可使用 ROLLFORWARD DATABASE 命令将日志应用于表空间，使其恢复到日志末尾。如果在进行崩溃恢复之前解决此问题，那么崩溃恢复操作足以使数据库恢复到一致并且可用的状态。

注意：

如果出错的表空间包含系统目录表，将不能启动数据库。必须复原 SYSCATSPACE 表

空间，然后执行前滚恢复直至日志末尾。

这种恢复结合使用完整的数据库备份和日志文件，从而扩展了版本恢复。在使用一个备份作为基准之前，必须首先存储这个备份，然后在该备份上应用日志。这个过程允许将数据库或表空间恢复到某个特定的时间点上。前滚恢复要求启用归档日志记录。

不同的恢复类型对日志的要求不同，下面我们讲解数据库日志。

7.2　DB2 日志

数据库备份是数据库的一个完整的副本。其实可以这样简单的理解：backup=copy，备份其实就是给数据库做一个特定时刻的 copy。那么恢复呢？恢复本质也是 copy 操作，恢复的本质就是恢复到我们备份时刻的数据。那么假设我们周一晚上做了一个数据库的备份，周二中午 12 点数据库存储介质出现了故障。如何能够实现恢复呢？那么我们首先是把数据库恢复到我们备份的那个时刻(也就是周一晚上)，但是备份之后和周二 12 点之间对数据库已经做的交易(事务)怎么办呢？这就需要用到数据库日志，因为一旦交易提交，我们对数据库做的 SQL 操作(insert、update 和 delete 等)都会记录到数据库日志中。所以我们就用数据库日志(前提是数据库日志没有受到损坏)把备份之后和数据库崩溃之前的所有 SQL 操作重做(redo)一遍。这就是数据库备份恢复的原理。所以大家可以看到在这个过程中，数据库日志至关重要。下面我们来详细讲解 DB2 日志。

7.2.1　日志文件的使用

为了确保用户数据的完整性，DB2 实施了提前写日志存档模式。提前写日志存档的基础是指：当发出删除、插入或更新数据库中某一数据的 SQL 调用时，所做出的数据变更首先要写到日志文件中去。当发出一条 SQL 提交确认命令时，DB2 要保证把为了重作(redo)所需要的日志文件都写入磁盘中。在发生断电之类的不幸事故时，日志文件可以用来把数据库退回到原来的某个一致性状态。所有被提交确认的事务都将重新再做一遍，所有未提交确认的事务都将退回到原有起点。图 7-5 展示了这种模式。

在图 7-5 中，一共执行了 4 条 SQL 语句。这些语句被缓存在程序包缓存中，数据页被从硬盘取出到缓冲池中。随着 SQL 语句的执行，更改首先被记录到日志缓冲区中，然后被写到日志文件中。在这个例子中，更改过的已经提交的数据页(脏页)还没有被写到硬盘上。这个工作通常是在需要腾出缓冲池空间时执行的，或者是由于性能的原因而后台异步(通过 db2pclnr 进程)地执行的。

DB2 Memory(in RAM)
Package cache
Update mytable set name='ZZZ'
where Name in('John', 'Raul')
Delete mytable
where Name = 'Peter'
Commit
Delete mytable
where Name = 'Mary'
Writing to
the Log Buffer
Log Buffer
Update Old value John/new value ZZZ
Update Old value Raul/new value ZZZ
Delete Old value = Peter
Commit
Delete Old value = Mary
Bufferpool
Data Page 1
Name = John
Name = Mary
Data Page 2
Name = Peter
Name = Raul
When the Log Buffer is full,
or a commit is issued,the
logs are written to the log
files.
Externalizing to the database
(in this example,it has not happend yet)
Name = John
Name = Mary
Name = Peter
Name = Raul
Name = WWW
Name = YYY
Disk for the database
Log File 1
Log File 2
Update Old value John/new value ZZZ
Update Old value Raul/new value ZZZ
Delete Old value = Peter
Commit
Delete Old value = Mary
Disk for logs

图 7-5 日志文件的使用

日志中只记录 DML 操作(insert、update 和 delete 操作)，假设我们在前台发出了一条 insert、update 或 delete 语句，那么在日志中就相应的记录这条 SQL 语句的 redo 操作和 undo 操作，undo 是为了执行回滚(rollback)操作;而 redo 是为了保证数据库可执行前滚(rollforward)操作来保证数据库的可恢复性。下面我们来看看一个 DB2 日志中主要记录哪些内容，请参阅表 7-1。

表 7-1 DB2 日志的内容

时间戳	LSN	用户名	运算	所有者	表	SQL 重做(redo)	SQL 还原(undo)
26-十月-2006 10:03:33	24814	PROD	DDL	PROD	T1	create table prod.t1(id int)	注：DDL 操作，create,drop 和 alter 操作不记录日志
26-十月-2006 10:03:33	24818	PROD	COMMIT			Commit	
26-十月-2006 10:03:45	24822	PROD	INSERT	PROD	T1	insert into "PROD"."T1" values "ID" = 1	delete from "PROD"."T1" where "ID" = 1 and RID = 'AAAGMNAAEAAABrHAAA'

(续表)

时间戳	LSN	用户名	运算	所有者	表	SQL 重做(redo)	SQL 还原(undo)
26- 十 月 -2006 10:04:00	24828	PROD	DELETE	PROD	T1	delete from "PROD"."T1" where "ID" = 1 and RID = 'AAAGMNAAEAAABrHAAA'	insert into "PROD"."T1" values "ID" = 1
26- 十 月 -2006 10:04:22	24836	PROD	UPDATE	PROD	T1	update "PROD"."T1" set "ID" = 200 where "ID" = 2 and RID = 'AAAGMNAAEAAABrHAAB'	update "PROD"."T1" set "ID" = 2 where "ID" = 200 and RID = 'AAAGMNAAEAAABrHAAB'
26- 十 月 -2006 10:04:23	24838	PROD	COMMIT			commit	

通过上面的日志我们可以看到，当我们在前台发出一个 insert、delete 或 update 语句时，日志中同时记录 redo 和 undo 操作，以保证数据库的前滚和回滚。大家注意前滚(rollforward)这个词和 redo 其实是等同的，回滚(rollback)和 undo 是等同的。

所有数据库都有与它们相关联的日志文件。日志文件有预先定义的大小(logfilsz)。因此，当日志文件被填满时，日志存档过程就要在另一个日志文件中继续进行。

7.2.2　日志类型

数据库中有两类日志：

- 循环日志(circular logging)
- 归档日志(archival logging)

1. 循环日志

循环日志记录是 DB2 默认的日志记录模式。顾名思义，这种类型的日志记录以循环的模式重用日志。例如，如果有 4 个主日志，DB2 将按照以下顺序使用它们：Log #1，Log #2，Log #3，Log #4，Log #1，Log #2，依此类推。

在循环日志记录模式下，只要这个日志包含已提交且已写到数据库磁盘上的操作，那么它就可以被重用。换句话说，如果日志仍然是活动日志，那么它就不能被重用。循环日志使用两类日志文件：

- 主日志文件(primary log files)
- 辅助日志文件(second log files)

主日志文件是预先分配的，而辅助日志文件仅在需要时才分配。如果数据库管理器需要序列中的下一个日志，并且主日志文件已经用完且不能被重用，那么将分配一个辅助日

志文件，直至主日志文件变得可供重用或者所分配辅助日志文件的数目被超出为止。一旦数据库管理器决定辅助日志文件不再需要时，辅助日志文件便被收回。当数据库被激活时，分配主日志文件，辅助日志仅当需要时才分配。

主日志文件和辅助日志文件的数目由数据库参数 LOGPRIMARY 和 LOGSECOND 来决定。

仍然使用前面循环日志记录的例子，如果有一个运行时间很长的事务，这个事务要横跨 5 个(图 7-6 中默认配置了 4 个主日志文件)日志文件，那么会出现什么情况呢？在这种情况下，DB2 会再多分配一个日志文件——即一个辅助日志文件。图 7-6 展示了其中的原理。

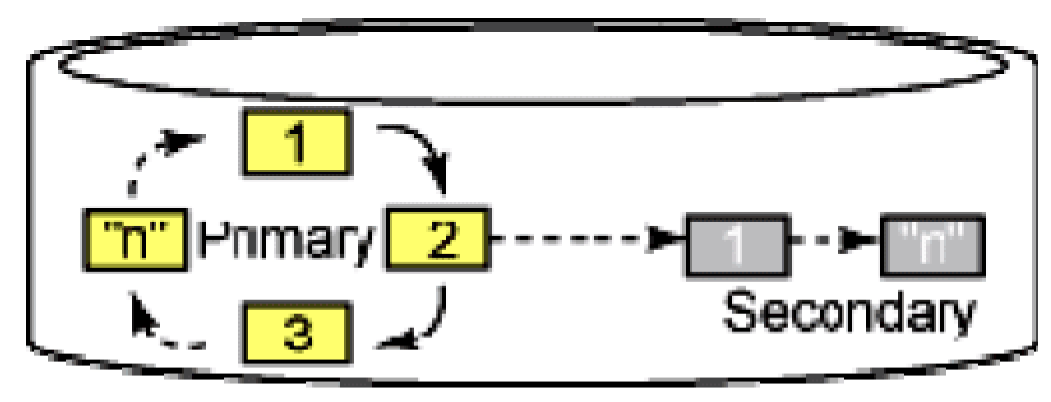


图 7-6　循环日志原理

当数据库最初被创建时，循环日志方式作为默认的日志方式被激活。

循环日志记录数据库仅能恢复到曾经做过备份的点。对数据库进行恢复时，备份点之后对数据库做的所有工作都将丢失。由于这个原因，循环日志方式最适合用于那些只供查询的数据库。

2. 归档日志

当使用归档日志记录模式时，您要经常归档(保留)日志。在循环日志记录模式下，已提交且被写到磁盘上的日志将被覆盖；而在归档日志记录模式下，这些日志将得到保留。例如，如果有 4 个主日志文件，DB2 将按照以下顺序使用它们：Log #1，Log #2，Log #3，Log #4(如果 Log #1 的所有事务已被提交且已写到磁盘上，则归档 Log #1)，Log #5(如果 Log #2 的所有事务已被提交且已写到磁盘上，则归档 Log #2)，Log #6，依此类推。

正如这个例子演示的那样，DB2 将保持 4 个主日志文件可用，即使一些日志文件中填满了已被提交且已写到磁盘上，DB2 也不会重用它们。DB2 不会覆盖已经成为归档日志的日志。图 7-7 阐释了其中的原理。

活动日志(由数 15 和 16 表示)

如果某个日志满足以下两个条件中任意一个，则属于活动(active)日志：

- 包含尚未提交或回滚的事务信息
- 包含已经提交但是更改尚未写到磁盘上的事务信息

注意：
活动日志特别重要，如果活动日志丢失，那么数据库将无法启动。

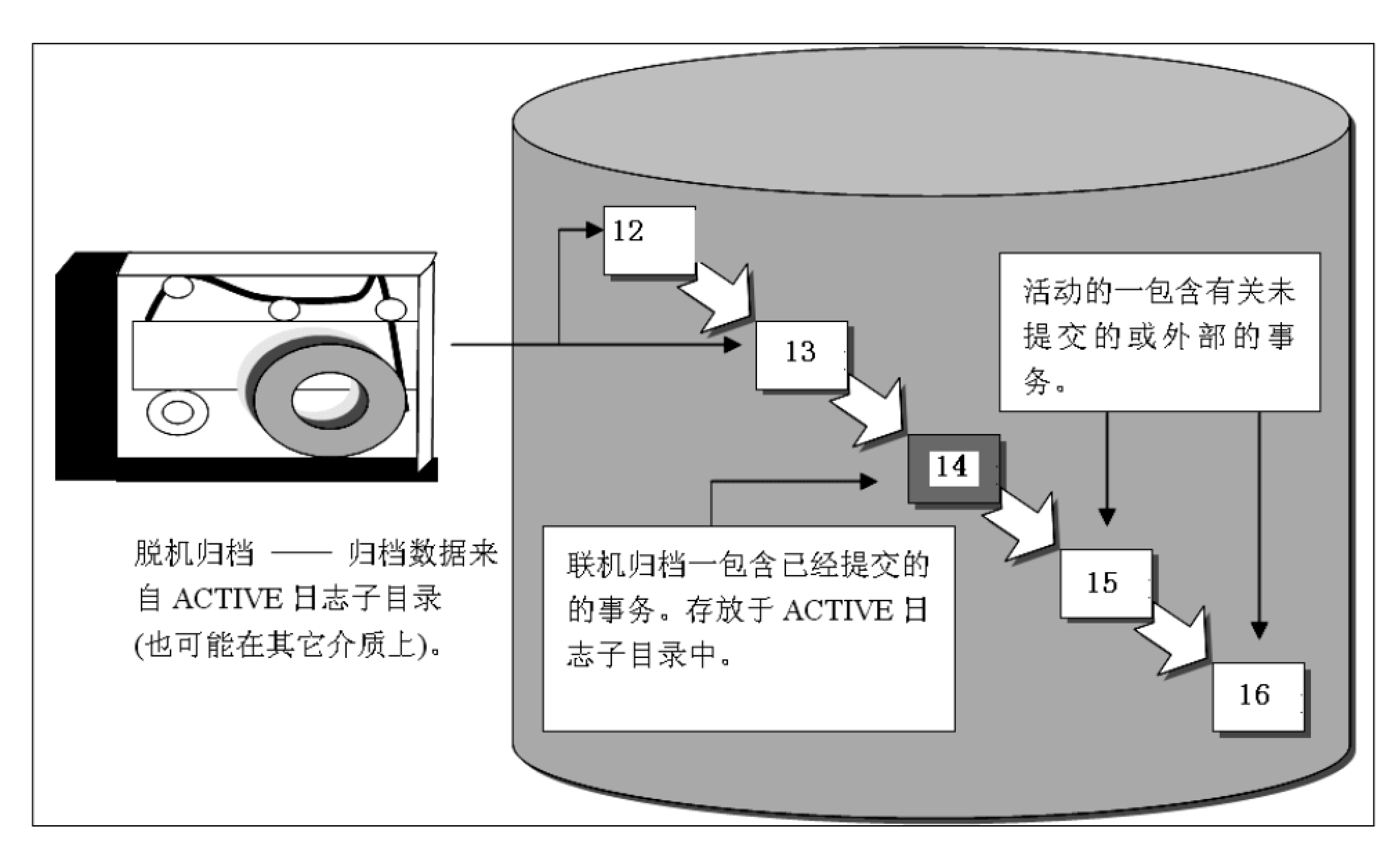


图 7-7　归档日志原理

在线归档日志(由数 14 表示)

包含已提交且已写到磁盘上的日志。这些日志与活动日志放在相同的目录中。

离线归档日志(由数 12 和 13 表示)

已经从活动日志目录转移到另一个目录或媒介上的归档日志。这种移动可以手动地完成，也可以由 DB2 自动完成。

一个数据库的日志记录的类型是由数据库参数 LOGARCHMETH1 决定的。当 LOGARCHMETH1 为 OFF(默认值)时，归档日志记录被禁用，循环日志记录被启用。

为了启用归档日志记录，可以将 LOGARCHMETH1 设置为以下值中的任何一个值：

- LOGRETAIN　日志文件将被保留在活动日志目录中
- USEREXIT　日志的归档和检索是由用户提供的用户出口程序自动执行的，这个出口程序必须由 db2uext2 调用。这个程序用于将在线归档日志移动到与活动日志目录不同的一个目录中，或者移动到另一个媒介上。当在 ROLLFORWARD 操作期间需要某些离线归档日志时，这个程序还可以用于将离线归档日志取出到活动日志目录中。在 Windows 下，db2uext2 必须存储在 sqllib\bin 目录中，在 UNIX 下，db2uext2 必须存储在 sqllib/adm 目录中
- DISK:directory_name　USEREXIT 使用与相同的算法。DB2 不调用用户出口程序，

而是自动将日志文件从活动日志目录归档到指定的目录

- TSM:[management class name]　使用与 USEREXIT 相同的算法。日志被归档到本地 Tivoli Storage Manger(TSM)服务器上。management class name 参数是可选的。如果没有指定该参数，则使用默认的管理类
- VENDOR:library_name　使用与 USEREXIT 相同的算法。日志是使用指定供应商的库来归档的

由于向后兼容的原因，数据库配置文件仍然包含参数 LOGRETAIN 和 USEREXIT。从 DB2 V8.2 开始，这两个参数已经被 LOGARCHMETH1 取代。如果更新 USEREXIT 或 LOGRETAIN 参数，那么 LOGARCHMETH1 将自动被更新，反之亦然。

归档日志方式不是默认的日志工作方式，但它是允许用户执行前滚(roll forward)恢复的唯一方法。

3. 无限日志记录

不管是使用循环日志记录还是归档日志记录，日志空间都可能被填满活动日志。如果启用无限日志记录，DB2 就会在一个日志被填满时立即归档这个日志。它不会等到日志中所有的事务都已经被提交且写到磁盘的时候才来归档日志。这样可以保证活动日志目录永远不会被填满。例如，如果有一个长时间运行的事务，在启用无限日志记录模式的情况下，就不会出现日志空间被耗尽的情况。

然而，我们不建议使用无限日志记录，因为它可能延迟紧急事故恢复的时间，这是因为需要从归档站点检索活动日志。无限日志记录是归档日志记录的一个派生物。要启用无限日志记录：

- 将 LOGSECOND 数据库配置参数设置为－1
- 启用归档日志记录

7.2.3　日志相关配置参数

DB2 的日志存档进程涉及到一系列参数。每当用户改变这些参数值时，用户都必须停止实例(db2stop)，然后再重新启动实例(db2start)，以便使得所读出的数据库配置带有变更后的参数数据。表 7-2 给出了影响日志工作的各个参数。

表 7-2　与日志记录相关的一些数据库配置参数

参　　数	用　　途
LOGPRIMARY	表明要分配的主日志文件的数量
LOGSECOND	表明可以分配的辅助日志文件的最多数量

(续表)

参　数	用　途
LOGFILSIZ	用于指定一个日志文件的大小(4KB 页的个数)
LOGBUFSZ	日志缓冲区大小参数决定分配多少内存空间用来作为缓冲区，在把日志记录写到磁盘之前暂时将它们保留在日志缓冲区。这一参数的值将表明 4KB 页的数目
NEWLOGPATH	日志文件默认的子目录定义在数据库目录的子目录 SQLLOGDIR 中。出于恢复目的的考虑，最好把日志文件存放到与数据库文件不同的物理磁盘中去。这一参数标识日志文件存放的新路径
SOFTMAX	软检查点，是个百分数，例如 50 表示日志文件写满 50%时，数据库执行 checkpoint 操作，把内存中的已经提交的数据(脏页)写到磁盘上
MINCOMMIT	组提交数，默认是 1，表示每次提交都写日志。假如设置为 5，则表示累计 5 次提交才写一次日志文件。如果没有累计到 5 次，那么每隔 1 秒写一次日志文件
LOGARCHMETH1	第一个日志归档方法
LOGARCHMETH1	第二个日志归档方法
MIRRORLOGPATH	镜像日志路径
TRACKMOD	启用增量备份

可以通过 update db cfg 命令来更改这些参数，例如如果启用归档日志，可以执行：

```
db2 update db cfg for sample using logretain recovery
```

在第一次启用归档日志时，需要给数据库做一个全备份。

如果要启用增量备份，可以执行：

```
db2 update db cfg for sample using logretain recovery trackmod on
```

更新归档日志文件目标文件夹(为归档日志文件指定路径可以将归档日志模式打开)：

```
db2 UPDATE DB CFG FOR db_name USING LOGARCHMETH1 “Disk:c:\backup”
```

7.2.4　数据库日志总结

日志记录类型与恢复类型

现在您理解了不同类型的日志记录和恢复类型，但是要注意的一点是，并不是所有日

志记录类型都支持所有的恢复类型。循环日志记录只支持崩溃恢复和版本恢复，而归档日志记录则支持所有类型的恢复：崩溃恢复、版本恢复和前滚恢复。

可恢复数据库和不可恢复数据库

可恢复(*recoverable*)数据库是指可以使用崩溃恢复、版本恢复或前滚恢复进行恢复的数据库；因此，对于这些数据库，需要启用归档日志记录。不可恢复(*nonrecoverable*)数据库是指不支持前滚恢复的那些数据库；因此，只需使用循环日志记录。

数据库日志总结

到目前为之，我们讲解了关于数据库日志和日志记录的一些概念。图 7-8 对这些概念作了总结。

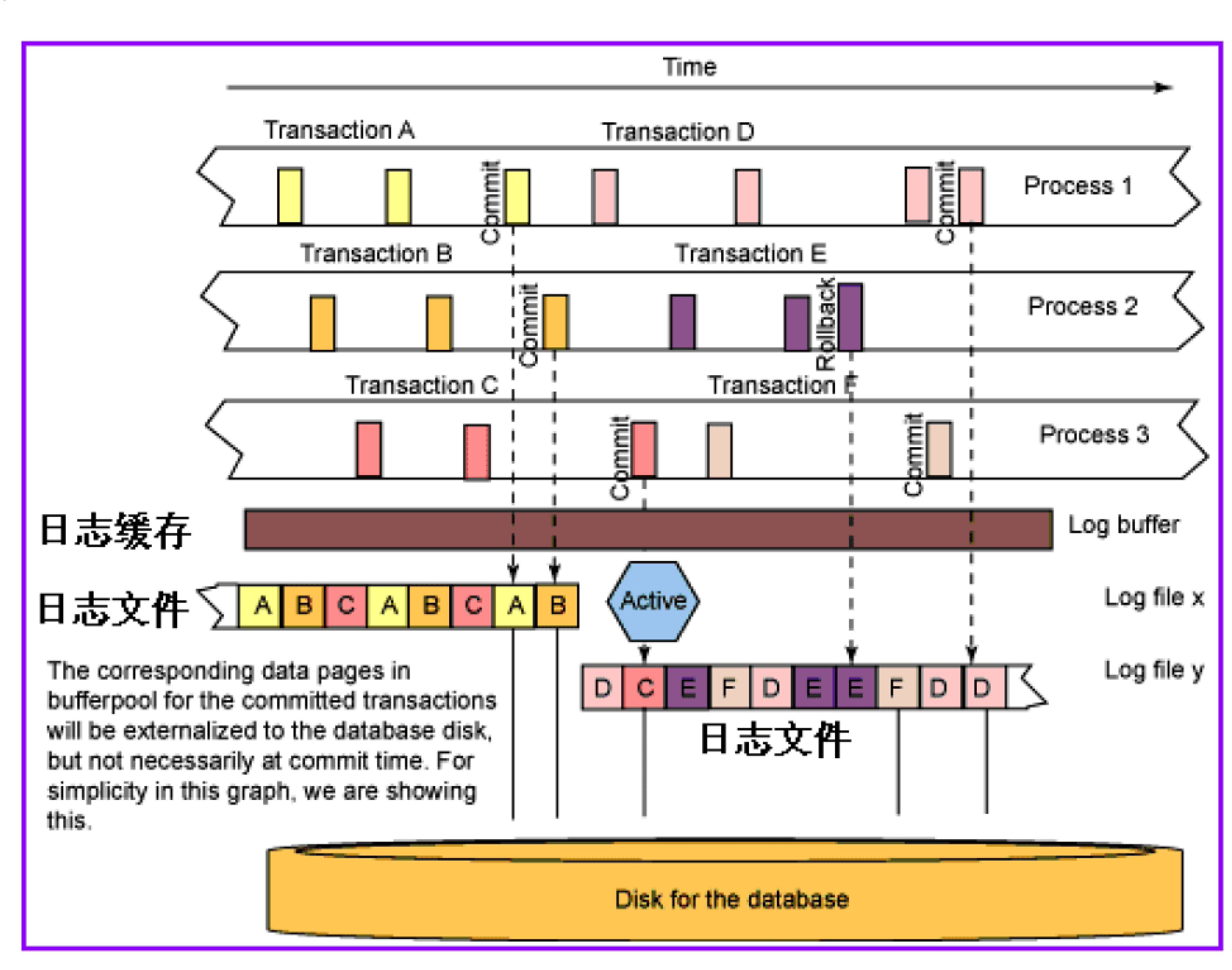


图 7-8　事务日志文件的使用

图 7-8 说明了如何使用多个日志文件去管理若干并行事务。图的顶部表示访问同一个数据库的三个用户进程(1-3)在时间上的进展。方框表示数据库的变更，比如插入或更新。用户可以看到每个事务(A-F)的生存期。图的中下方表示数据库的变更如何被同步地记录到日志文件(x，y)中。每个小方格中的字母表示该数据库变更属于哪个事务。

当发出一条 SQL COMMIT 命令或日志缓冲区被填满时，包含该事务的日志缓冲区便被写入到磁盘。事务 E 任何时候都不被写入磁盘，因为它用 SQL ROLLBACK 命令来终止。当日志文件 x 因存放事务 B 的第一个数据库变更而用完了空间时，日志存档进程将转向日志文件 y 继续进行。日志文件 x 仍然保持活动状态，直到把事务 C 的所有变更都写入到磁盘中为止。在将日志存档工作转向日志文件 y 的瞬间，日志文件 x 仍然保持处于活动状态的时间周期，用一个小的六边形来表示。

标有 *active* 字样的六边形表示日志文件 x 仍然被当作活动日志的时间。您可以看到，这个六边形在表示日志文件 y 中事务 D 和事务 C 一部分的正方形之上。为什么日志文件 x 在被填满之后仍然被当作活动日志呢？这是因为它包含了还没有被提交和写到磁盘上的日志信息。日志文件 x 包含事务 A、B 和 C。只有事务 A 和事务 B 已经被提交(在这个例子中，它们也随之立即被写到磁盘上)；而事务 C 仍然在运行，并且被写到日志文件 y 中。当事务 C 在日志文件 y 中被提交时(在这个例子中，它也随之立即被写到磁盘上)，日志文件 x 将不再被当作活动日志，它将成为一个在线归档日志。

最好大家要记住日志中记录的是我们对数据库做的操作(insert、update 和 delete 等)，而表空间对应的容器上记录的是我们真正的数据。有人经常问我，日志和数据之间是什么样的增长关系。我在这里给大家举一个简单的例子：假设有一个表，表中有一个 INT 字段 seqno，我把这个字段 update 了 1000 次，这个表的数据并没有真正的变化。但是数据库日志中会记录 1000 次 update 操作。所以日志和表数据量大小没有必然的关系，日志只和你做的 SQL 操作的多少有关系。

7.3　数据库和表空间备份

在线访问与离线访问

如果我们在执行一个在线操作(备份、恢复、复原)，那么其他用户也可以同时访问我们正在操纵的数据库对象。如果我们在执行一个离线操作，那么就不允许任何其他用户同时访问我们正在操纵的数据库对象。在这一节中，我们会经常使用术语“在线”和“离线”。

7.3.1　数据库备份

数据库备份是数据库的一个完整的副本。其实可以这样简单地理解：backup=copy，备份其实就是给数据库做一个特定时刻的 copy。除了数据外，备份副本还包含关于表空间、容器、数据库配置、日志控制文件和恢复历史文件的信息。注意，备份不会存储数据库管理器配置文件或注册表变量。只有数据库配置文件才会得到备份。

要执行备份，需要 SYSADM、SYSCTRL 或 SYSMAINT 权限。

下面是用于备份的 BACKUP 命令实用程序的语法：

```
BACKUP DATABASE database-alias [USER username [USING password]]
[TABLESPACE (tblspace-name [ {,tblspace-name} ... ])] [ONLINE]
[INCREMENTAL [DELTA]] [USE {TSM | XBSA} [OPEN num-sess SESSIONS]
[OPTIONS {options-string | options-filename}] | TO dir/dev
[ {,dir/dev} ... ] | LOAD lib-name [OPEN num-sess SESSIONS]
[OPTIONS {options-string | options-filename}]]
[WITH num-buff BUFFERS] [BUFFER buffer-size] [PARALLELISM n]
[COMPRESS [COMPRLIB lib-name [EXCLUDE]] [COMPROPTS options-string]]
[UTIL IMPACT PRIORITY [priority]] [{INCLUDE | EXCLUDE} LOGS] [WITHOUT
PROMPTING]
```

让我们来看看一些例子，以了解其中一些选项的作用。

要为数据库 sample 执行完整的离线备份，并将备份副本存储在 d:\backup 目录中，可以使用以下命令：

```
BACKUP DATABASE sample TO d:\backup
```

要使用其他选项为数据库 sample 执行完整离线备份，可以使用以下命令：

```
(1)   BACKUP DATABASE sample
(2)   TO /db2backup/dir1, /db2backup/dir2
(3)   WITH 4 BUFFERS
(4)   BUFFER 4096
(5)   PARALLELISM 2
```

我们来更仔细观察前面的命令：

(1) 表明要备份的数据库的名称(或别名)。

(2) 指定用于存储备份的位置，可以指定多个路径，每个路径下的备份文件会有一个序列号，例如 001、002 等。

(3) 表明在备份操作期间可以使用多少个内存缓冲区。使用多个缓冲区可以提高性能。

(4) 表明每个备份缓冲区的大小。

(5) 决定使用多少并行读/写进程/线程来进行备份。

对于 DB2 V9，建议大多数选项使用默认值，因为备份实用程序自己可以自动决定使用什么选项能获得最佳性能。

语法中没有关键字 OFFLINE，因为这是默认模式。如果要为 sample 数据库执行完整在线备份，则必须指定关键字 ONLINE。在线备份要求启用数据库的归档日志记录。下面的语句展示了如何执行在线备份：

```
BACKUP DATABASE sample  ONLINE  TO C:\backup
```

由于在线备份允许用户在执行备份的同时访问数据库，因此这些用户作出的更改很可能不会包含在备份副本中。因此，仅仅凭借在线备份还不足以进行恢复，另外还需要备份操作期间收集到的相应的日志。当在线备份完成时，DB2 强制关闭当前活动日志(并将其归档)，因此很容易在备份完成时收集当前活动日志。

为了将日志也备份到备份副本中，可以使用 BACKUP DATABASE 命令的 INCLUDE LOGS 选项。这样可以确保即使丢失了日志，也仍然可以使用备份镜像中包含的日志来恢复到最小的时间点上。INCLUDE LOGS 选项只适用于在线备份。

例如，要对 sample 数据库和日志进行在线备份，并以 D:\backup 作为目标目录，可以发出：

```
BACKUP DATABASE sample ONLINE TO D:\backup INCLUDE LOGS
```

7.3.2　表空间备份

如果一个数据库中只有一部分表空间有较多的变更，那么可以选择不备份整个数据库，而是只备份特定的表空间。

要执行表空间备份，可以使用以下语法：

```
BACKUP DATABASE sample
TABLESPACE ( syscatspace, userspace1, userspace2 )
ONLINE
TO /db2tbsp/backup1, /db2tbsp/backup2
```

上面例子中的第 2 行表明这是一个表空间备份，而不是完整数据库备份。还应注意，您可以在备份中包括任意多个表空间。临时表空间不能使用表空间级备份进行备份。

通常，您希望将相关的表空间备份在一起。例如，假设您正在使用 DMS 表空间，其中一个表空间用于表数据，另一个表空间用于索引，还有一个表空间用于 LOB。那么您应该同时备份这几个表空间，以便拥有一致的信息。对于所包含的表之间定义了参照约束的表空间，也应该如此。

7.3.3　增量备份

要执行增量备份，必须首先把 DB 配置参数 trackmod 设置为 ON，有两种类型的增量备份：

- **增量备份：**DB2 备份自上一次完整数据库备份以来发生变化的所有数据。

- **delta 备份：**DB2 备份自上一次成功执行的完整备份、增量备份或 delta 备份以来发生变化的数据。

图 7-9 展示了这两种类型的备份之间的不同之处。

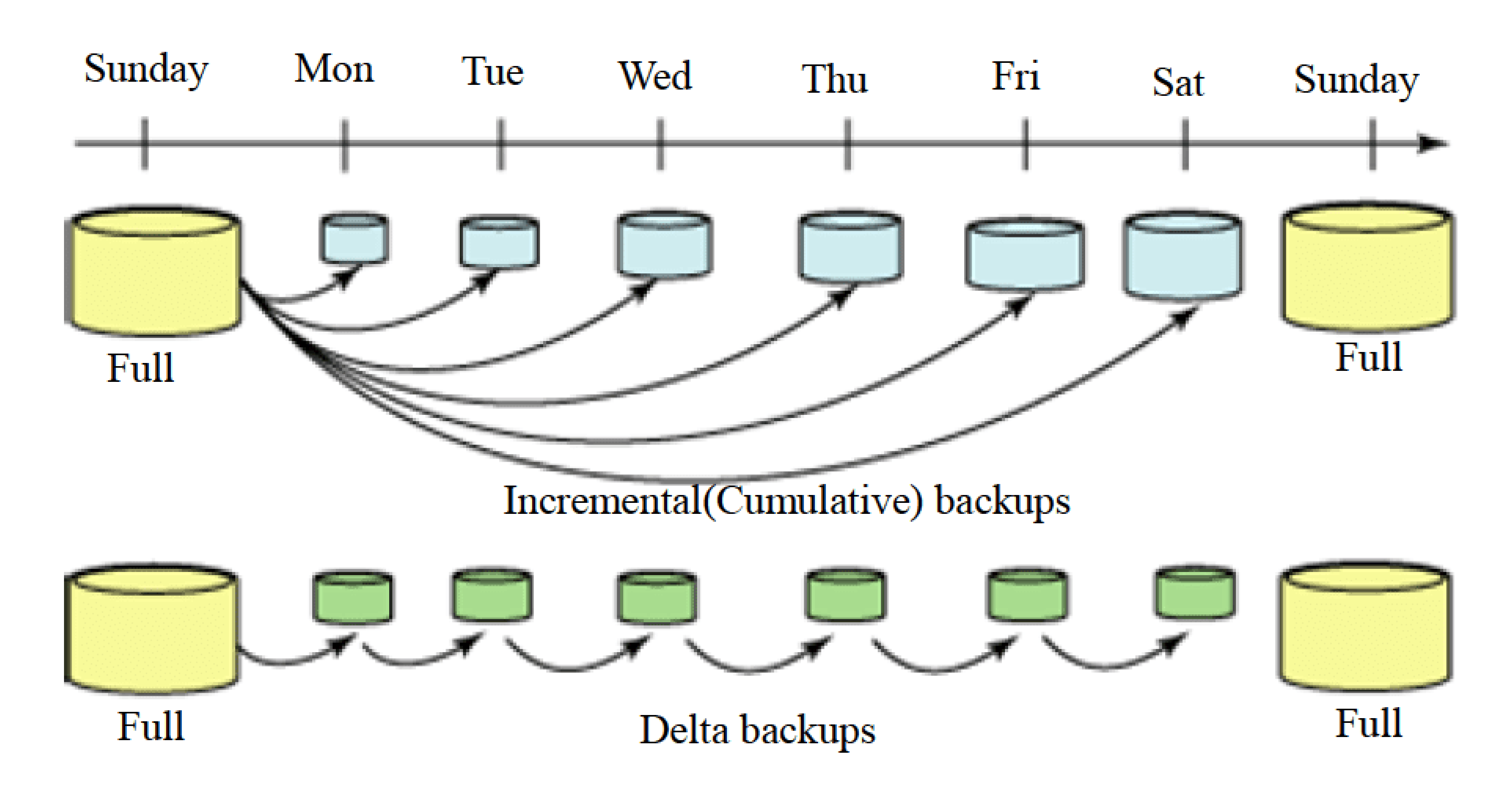


图 7-9　增量备份和 delta 备份间的差异

在图 7-9 的上端，如果在星期五执行了增量备份之后系统出现了崩溃，那么可以首先恢复第一个星期天的完整备份，然后再恢复星期五做的增量备份。要执行增量(累积)备份，可以执行如下命令：

```
BACKUP DATABASE sample incremental TO /db2tbsp/backup1, /db2tbsp/backup2
```

其中 incremental 表示我们执行的是增量(累积)备份。

在图 7-9 的下端，如果在星期五执行了 delta 备份之后系统出现了崩溃，那么可以首先恢复第一个星期天的完整备份，然后再恢复从星期一到星期五做的每个 delta 备份。要执行 delta(差分)备份，可以执行如下命令：

```
BACKUP DATABASE sample incremental delta TO /db2tbsp/backup1, db2tbsp/backup2
```

其中 incremental delta 表示我们执行的是增量 delta(差分)备份。

使用控制中心执行备份

除了使用命令行外，我们还可以使用控制中心来执行备份。图 7-10 展示了如何从控制中心调用 BACKUP 实用程序。要执行数据库备份或表空间备份，可以在要备份的数据库上单击右键，并选择“备份”，如图 7-10 所示。

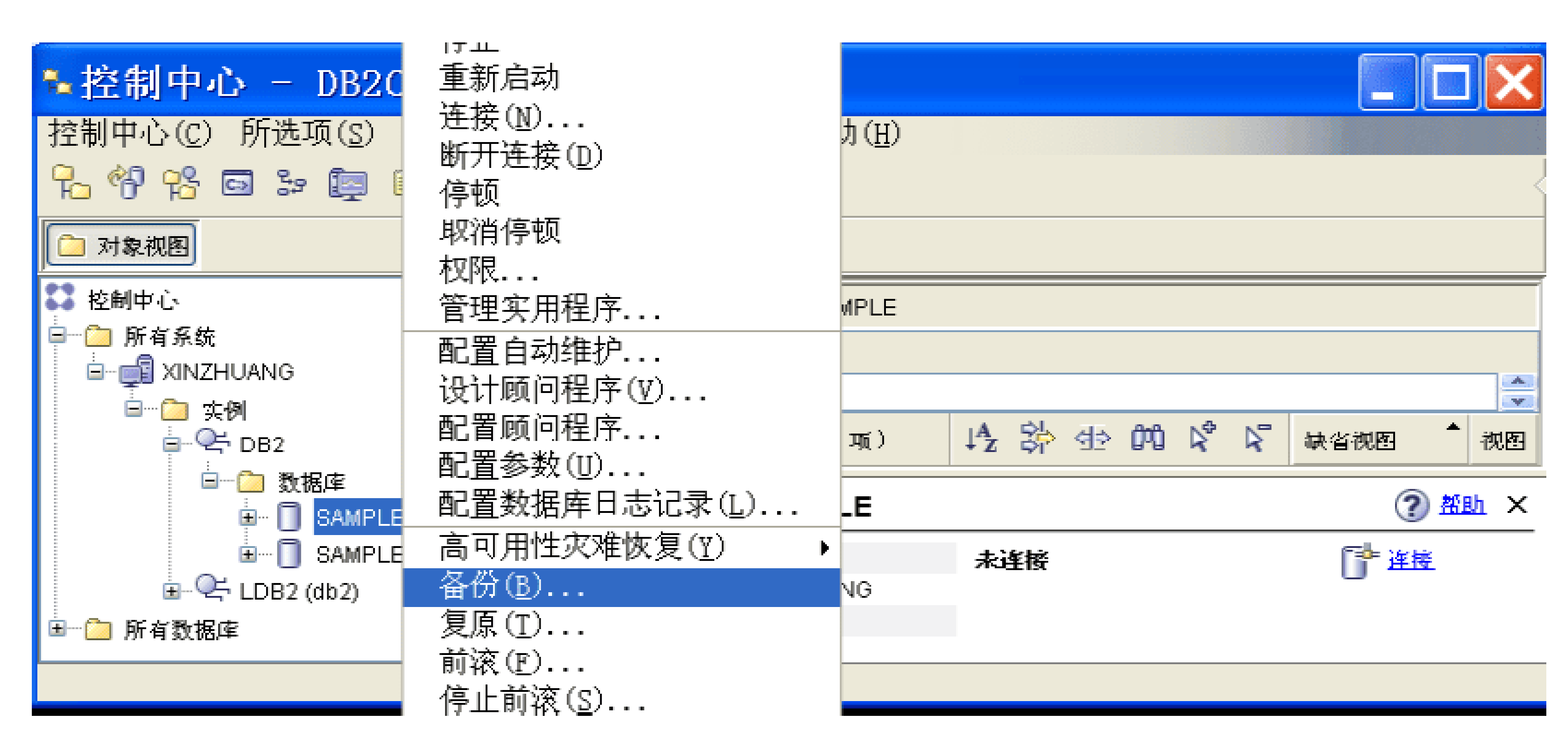


图 7-10　选择要备份的数据库

接下来的图 7-11 展示了执行 BACKUP 实用程序所需的“备份向导”和选项。

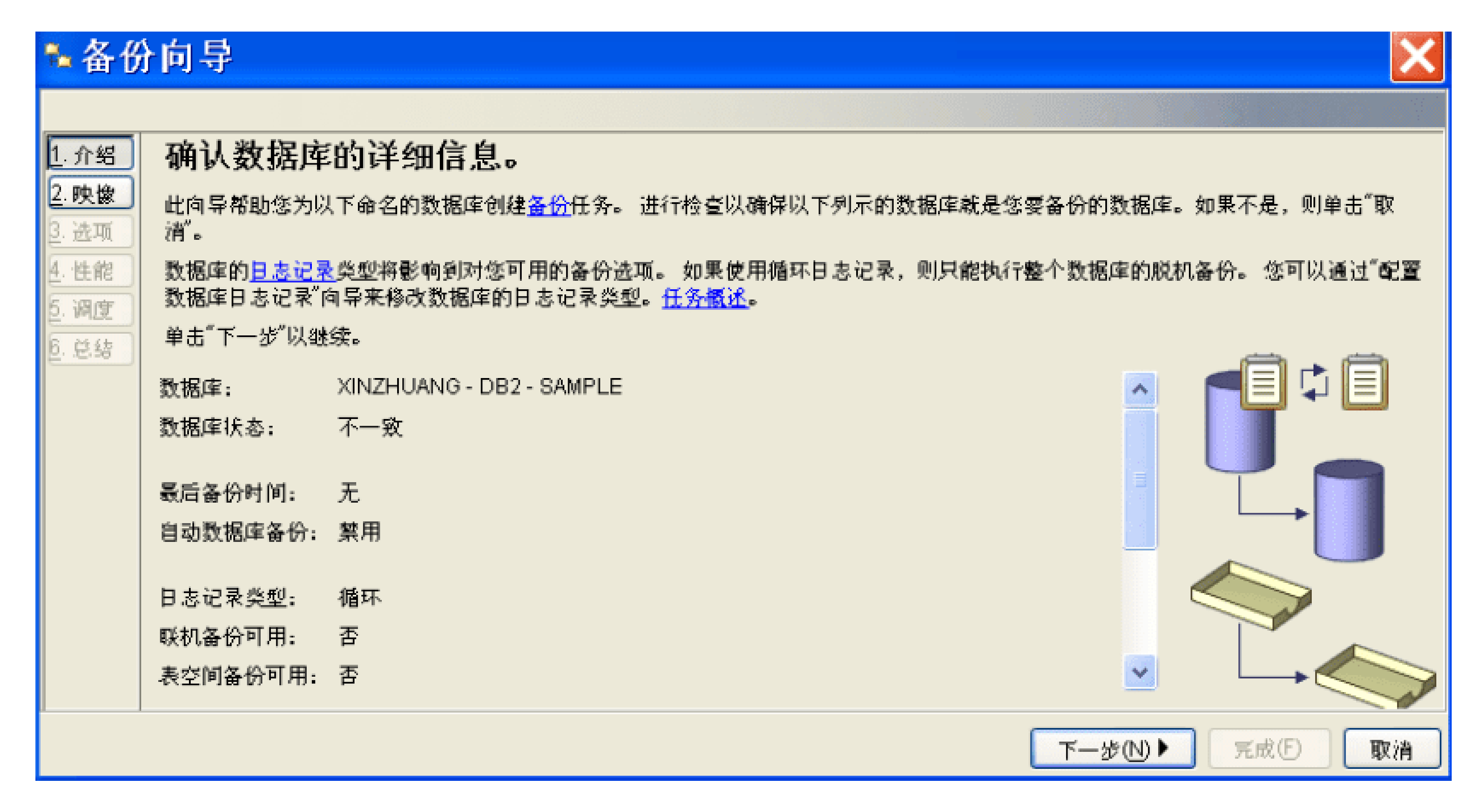


图 7-11　备份向导

备份文件的命名惯例

DB2 备份文件的命名惯例包括以下几项：

- 数据库别名
- 表示备份类型的数字(0 表示完整数据库备份，3 表示表空间备份，4 表示来自 LOAD 的副本)
- 实例名称
- 数据库节点(对于单分区数据库，总是为 NODE0000)
- 编目节点号(对于单分区数据库，总是为 CATN0000)
- 备份的时间戳
- 镜像序列号

图 7-12 展示了上述命名惯例，它适用于所有平台(在 DB2 V9 之前，备份文件在 Windows

平台上是类似 C:\SAMPLE.0\DB2INST.0\20090414\131259.001 的格式)。

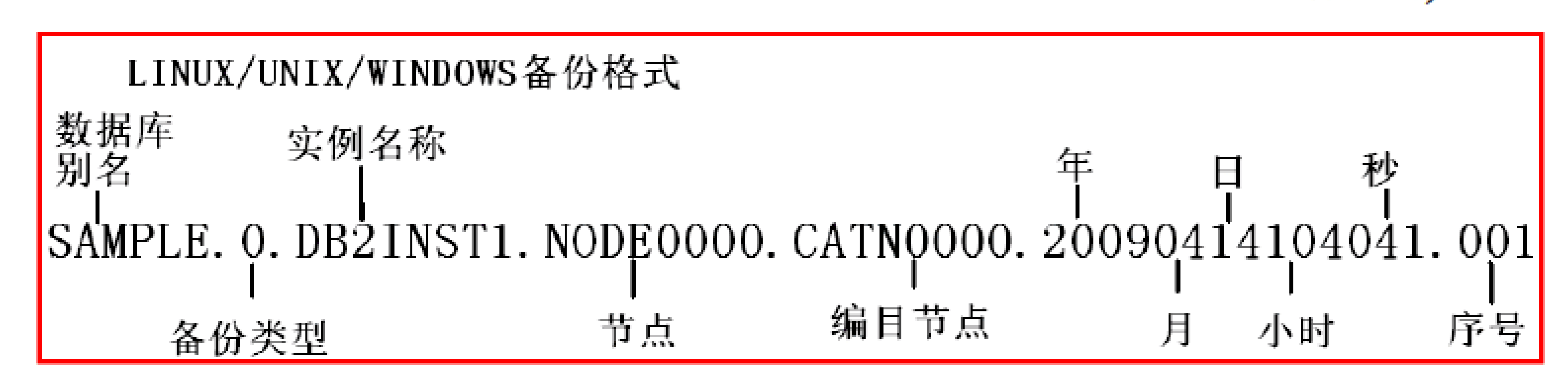


图 7-12　DB2 备份文件的命名惯例

7.3.4　检查备份完整性——db2ckbkp

当我们备份完成后，使用 **db2ckbkp** 命令不仅可以用来检查 DB2 数据库备份文件的完整性，而且还可以用来查询 DB2 数据库备份文件的元数据。如果我们有一些备份文件，但是不知道备份的类型，我们可以使用 **db2ckbkp -h** <备份文件>来检查 DB2 数据库备份的类型。

例如在我们的目录中有如下几个备份文件：

```
    C:\>dir SAMPLE*
     驱动器 C 中的卷没有标签。
     卷的序列号是 2CF5-258F
     C:\的目录
    2008-11-19  19:31      146,735,104
SAMPLE.0.DB2.NODE0000.CATN0000.20081119193103.001
    2008-11-19  23:25      201,326,592
SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001
                2 个文件   348,061,696 字节
                0 个目录  7,538,397,184 可用字节
```

我们想检查一下文件 SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001 的备份类型，可以使用如下命令：

```
    db2ckbkp -h SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001
    C:\>db2ckbkp -h SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001
    =====================
    MEDIA HEADER REACHED:
    =====================
            Server Database Name           -- SAMPLE
            Server Database Alias          -- SAMPLE
            Client Database Alias          -- SAMPLE
            Timestamp                      -- 20081119232532
            Database Partition Number      -- 0
            Instance                       -- DB2
            Sequence Number                -- 1
```

```
          Release ID                          -- C00
          Database Seed                       -- 491AA028
          DB Comment's Codepage (Volume) -- 0
          DB Comment (Volume)                 --
          DB Comment's Codepage (System) -- 0
          DB Comment (System)                 --
          Authentication Value                -- 255
          Backup Mode                         -- 0
          Includes Logs                       -- 0
          Compression                         -- 0
          Backup Type                         -- 0
          Backup Gran.                        -- 0
          Status Flags                        -- 11
          System Cats inc                     -- 1
          Catalog Partition Number            -- 0
          DB Codeset                          -- UTF-8
          DB Territory                        --
          LogID                               -- 1221200288
          LogPath                         -- C:\DB2\NODE0000\SQL00001\SQLOGDIR\
          Backup Buffer Size                  -- 3411968
          Number of Sessions                  -- 1
          Platform                            -- 5
     The proper image file name would be:
    SAMPLE.0.DB2.NODE0000.CATN0000.20081119232532.001
    [1] Buffers processed:
############################################################
    Image Verification Complete - successful.
```

我们可以通过上述输出中的 Backup Mode，Backup Type 和 Backup Gran.来确定备份的类型，三个关键字的说明如下：

- Backup Mode 0 - offline(脱机备份)，1 - online(联机备份)
- Backup Type 0 - full(全备份)，3 - tablespace(表空间级备份)
- Backup Gran. 0 - normal(正常备份)，16 - incremental(增量备份)

48 - delta(增量 delta 备份)

其中 incremental(增量备份)和 delta(增量 delta 备份)的说明如下：

◊ incremental(增量备份)：增量备份是最近一次成功的完全备份时点之后的数据库操作的副本，因为每个增量备份都包含上一次增量备份的内容，所以也称为累积备份。最近一次成功的完全备份是增量备份的基础。

◊ delta(增量 delta 备份)：delta 备份映像或增量 delta 备份映像是自从上次数据库的成功备份(包括完整、增量或 delta 备份)以来，已更改过的所有数据库数据的

副本。也称为差异备份映像或非累积备份映像。delta 备份映像的前身是最新的成功备份，包括 delta 备份映像中每个数据库备份。

- Includes Logs：表示在线备份期间是否包含备份期间产生的数据库日志。0 为否，1 为是。
- Compression：是否启用备份压缩。0 表示没有启动，1 表示启用备份压缩。

明确了上述数字的含义后，我们就可以很容易地辨别上述备份文件是否属于“联机全备份”：

- Backup Mode -- 1 (联机备份)
- Backup Type -- 0 (全备份)
- Backup Gran. -- 0 (正常备份)

如果要检查磁带上数据库备份的完整性，可以使用 db2ckbkp –n 选项。

7.4 数据库和表空间恢复

7.4.1 数据库恢复

本节讨论 RESTORE 实用程序，该实用程序使用一个备份文件作为输入，输出是一个新的或已有的数据库。虽然我们讨论的是数据库恢复(recovery)，但是我们使用的实用程序是 RESTORE，而不是 RECOVER。

要恢复到已有的数据库，需要 SYSADM、SYSCTRL 或 SYSMAINT 权限。要恢复到新的数据库，则需要 SYSADM 或 SYSCTRL 权限。

下面是 RESTORE 命令的语法：

```
    RESTORE DATABASE source-database-alias { restore-options | CONTINUE | ABORT }
    restore-options:
    [USER username [USING password]]
    [REBUILD WITH [ALL TABLESPACES IN DATABASE | ALL TABLESPACES IN IMAGE]
       [EXCEPT rebuild-tablespace-clause] [rebuild-tablespace-clause]
    [{TABLESPACE [ONLINE] | TABLESPACE (tblspace-name [ {,tblspace-name} .. ])
[ONLINE] |
    HISTORY FILE [ONLINE]}] [INCREMENTAL [AUTOMATIC | ABORT]]
    [{USE {TSM | XBSA} [OPEN num-sess SESSIONS] |
    FROM dir/dev [ {,dir/dev} ... ] | LOAD shared-lib
    [OPEN num-sess SESSIONS]}] [TAKEN AT date-time] [TO target-directory]
    [INTO target-database-alias] [NEWLOGPATH directory] [LOGS FROM directory]
       [LOGTARGET directory]
    [WITH num-buff BUFFERS] [BUFFER buffer-size]
```

```
[DLREPORT file-name] [REPLACE EXISTING] [REDIRECT] [PARALLELISM n]
[WITHOUT ROLLING FORWARD] [WITHOUT DATALINK] [WITHOUT PROMPTING]
Rebuild-tablespace-clause:
[TABLESPACE (tblspace-name [ {,tblspace-name} ... ])
```

让我们看一个例子。要执行 sample 数据库的恢复，可以使用以下命令：

```
(1)  RESTORE DATABASE sample
(2)  FROM  C:\DBBACKUP
(3)  TAKEN AT 20080314131259
(4)  WITHOUT ROLLING FORWARD
(5)  WITHOUT PROMPTING
```

在上面的例子中：

(1) 表明要恢复的数据库镜像的名称。

(2) 指定要从中读取输入备份文件的位置。

(3) 如果该目录中有多个备份镜像，该选项将基于时间戳(备份名称的一部分)标识特定的备份。

(4) 如果一个数据库启用了归档日志记录，那么当该数据库被恢复时，它将自动被置于 rollforward pending 状态。这一行告诉 DB2 不要将该数据库置于 rollforward pending 状态。

注意：

在下列情况下，不能使用 WITHOUT ROLLING FORWARD 选项：

- 正在从联机备份映像复原
- 正在发出表空间级复原

(5) 当执行 RESTORE 时，您将看不到任何提示。

注意，以上语法中没有关键字 OFFLINE，因为这是默认模式。对于 RESTORE 实用程序来说，这是能用于数据库的唯一模式。

如果备份镜像中包括日志文件，那么可以使用 RESTORE DATABASE 命令的 LOGTARGET 选项恢复日志文件。

为了使用 C:\DBBACKUP 目录中的备份镜像恢复 SAMPLE 数据库，并将日志文件恢复到 C:\DB2\NODE0000\SQL00001\SQLOGDIR 目录，可以发出：

```
RESTORE DATABASE sample FROM C:\DBBACKUP
LOGTARGET C:\DB2\NODE0000\SQL00001\SQLOGDIR
```

也可以通过使用 LOGS 关键字只恢复日志文件，而不恢复数据库：

```
RESTORE DATABASE sample LOGS FROM C:\DBBACKUP
```

```
LOGTARGET C:\DB2\NODE0000\SQL00001\SQLOGDIR
```

7.4.2 表空间恢复

可以使用完整数据库备份或表空间备份来恢复表空间。对于表空间恢复，要小心地做好计划，因为一不小心就会使数据处于不一致状态。

下面是表空间 RESTORE 命令的一个例子：

```
(1)  RESTORE DATABASE sample
(2)  TABLESPACE ( mytblspace1 )
(3)  ONLINE
(4)  FROM /db2tbsp/backup1, /db2tbsp/backup2
```

在上面的例子中：

(1) 表明要恢复的数据库镜像。

(2) 表明这是表空间 RESTORE，并指定要恢复的表空间的名称。

(3) 表明这是在线恢复。对于用户表空间，在线恢复和离线恢复都是允许的。如前所述，对于数据库，只允许离线恢复。

(4) 指定输入备份文件所在的位置。

1. 关于表空间恢复的考虑

表空间被恢复之后，它总是被置于 rollforward pending 状态。为了使表空间可以被访问，并重置这个状态，必须至少将表空间前滚一个最小的时间点。这个最小的时间点可以确保表空间和日志与系统编目表中的信息一致。

考虑下面这个例子：

(1) 假设在时间 t1 上您做了一个完整数据库备份，其中包括表空间 mytbls1。

(2) 在时间 t2 上，您在表空间 mytbls1 中创建了表 myTable。这将为表空间 mytbs1 到 t2 的恢复设置最小时间点。

(3) 在时间 t3 上，您决定使用 t1 时做的完整数据库备份只恢复表空间 mytbls1。

(4) 恢复完成之后，表空间 mytbls1 将被置于 rollforward pending 状态。如果您可以前滚到最小时间点之前的一个时间点上，那么表空间 mytbls1 中将不包括表 myTable；然而，系统编目却说这个表的确在 mytbls1 中。为了避免出现这样的不一致，当您恢复一个表空间时，DB2 将强制使您至少前滚到最小时间点上。

当对表空间或表空间中的表运行 DDL 语句时，最小时间点将被更新。为了确定一个表空间的恢复的最小时间点，可以使用以下两种方法之一：

- 使用 LIST TABLESPACES SHOW DETAIL 命令

- 使用 GET SNAPSHOT FOR TABLESPACE ON db_name 命令获得表空间快照

系统编目表空间(SYSCATSPACE)也必须前滚到日志的最后，并处于离线模式。在 7.5 节我们将更详细地讨论 ROLLFORWARD 命令。

2. 使用控制中心执行恢复

除了可以使用命令行恢复数据库和表空间外，我们还可以使用控制中心进行恢复。图 7-13 展示了如何从控制中心调用 RESTORE 实用程序。要执行数据库或表空间恢复，可以在要恢复的数据库上单击右键，并选择“**复原**”。

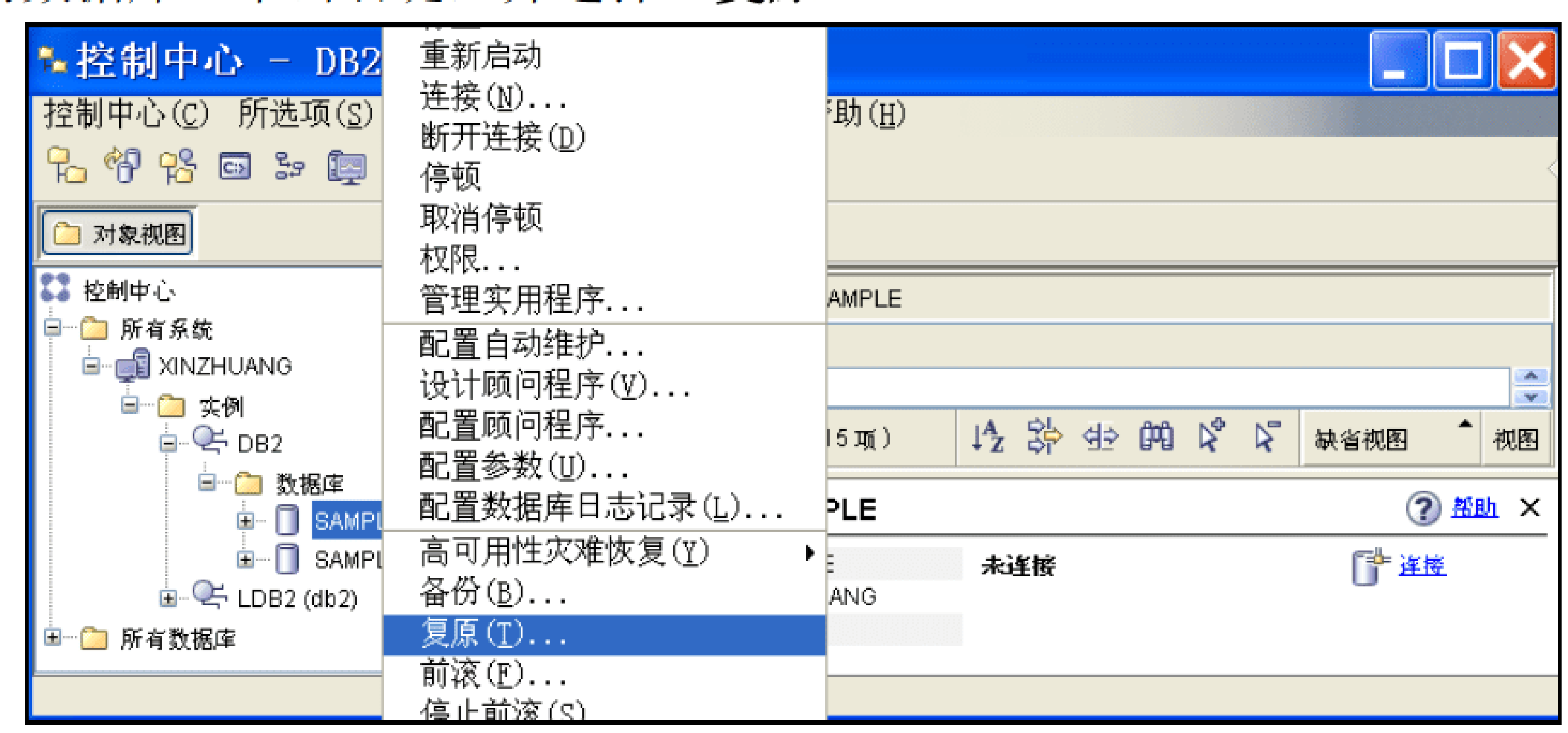


图 7-13　从控制中心调用 RESTORE 实用程序

接下来的图 7-14 展示了执行 RESTORE 实用程序所需的一些选项。

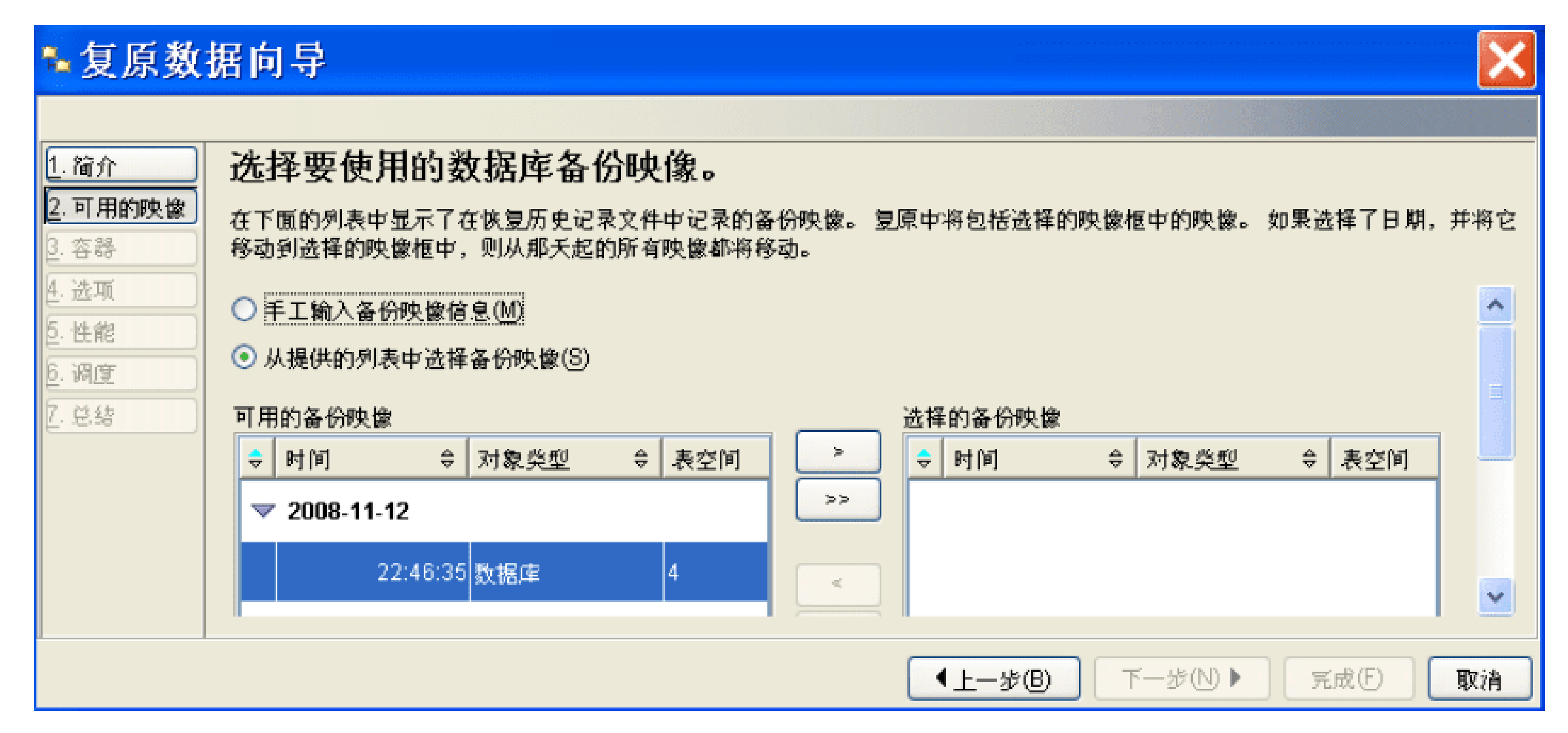


图 7-14　执行 RESTORE 实用程序所需的一些选项

7.4.3　增量恢复

还记得吗，增量备份是只包含自从执行前一个备份以来更新过的页的备份镜像。要使用增量备份来执行恢复，可以在 RESTORE DATBASE 命令上指定 INCREMENTAL 选项。

通常，增量恢复会涉及到一系列的恢复操作。恢复中涉及的每个备份镜像必须按以下顺序进行恢复：最近一次，第一次，第二次，第三次，依此类推，直到最后一个镜像。

让我们看一个例子。下面是用于可恢复数据库 MYDB 的一个每周增量备份策略。它包括一个每周完整数据库备份操作、一个每日差分(delta)备份操作和每周中期的累积(增量)备份操作：

```
1. (Sun) backup db mydb to c:\backup
2. (Mon) backup db mydb online incremental delta to c:\backup
3. (Tue) backup db mydb online incremental delta to c:\backup
4. (Wed) backup db mydb online incremental to c:\backup
5. (Thu) backup db mydb online incremental delta to c:\backup
6. (Fri) backup db mydb online incremental delta to c:\backup
7. (Sat) backup db mydb online incremental to c:\backup
```

为了使用星期五创建的增量备份恢复数据库，需要发出一系列的 RESTORE DATABASE 命令来恢复所需的每个镜像，顺序如下：

```
1. restore db mydb incremental taken at (Fri)----时间戳
2. restore db mydb incremental taken at (Sun)
3. restore db mydb incremental taken at (Wed)
4. restore db mydb incremental taken at (Thu)
5. restore db mydb incremental taken at (Fri)
```

您可以看到，这个过程非常冗长，而且也容易出错。您必须知道要恢复的镜像是什么，还要知道以什么顺序恢复这些镜像。为了使这个过程变得更容易，可以使用自动增量恢复实用程序。为了执行自动增量恢复，只需识别出要恢复的最近的备份镜像，然后对它发出一个带 INCREMENTAL AUTOMATIC 选项的 RESTORE DATABASE 命令即可。恢复实用程序将负责剩下的事情。

上述所有手动恢复命令都可以放在一个自动增量恢复命令中：

```
restore db mydb incremental automatic taken at (Fri)
```

7.4.4 增量恢复检查——db2ckrst

对于 DB2 数据库来说，其非增量备份镜像的恢复操作是可以通过仅发出两条恢复命令来完成的。但如果要恢复数据库的增量备份镜像，我们可以使用 db2ckrst 这一检测增量备份镜像恢复顺序的实用程序检测要恢复的增量备份镜像的时间戳记，以获取恢复操作的命令序列。下面是一个例子：

(1) 获取恢复操作的命令序列：

```
SAMPLE.0.DB2INST1.NODE0000.CATN0000.20080429143406.001;
 db2ckrst -d sample -t 20080429143406
```

(2) 命令执行后，在屏幕上会有类似的如下输出：

```
Suggested restore order of images using timestamp 20080429143406 for
database sample.
=============================================
restore db sample incremental taken at 20080429143406
restore db sample incremental taken at 20080429142824
restore db sample incremental taken at 20080429143406
=============================================
```

(3) 恢复增量备份镜像：

```
db2 restore db sample incremental taken at 20080429143406
db2 restore db sample incremental taken at 20080429142824
db2 restore db sample incremental taken at 20080429143406
```

7.4.5　重定向恢复

之前我们提到，备份文件包括表空间和容器的信息。如果在做备份时还存在的一个容器恢复时已经不存在了，那么会出现什么情况呢？如果 RESTORE 实用程序不能发现这个容器，那么就会碰到一个错误。

如果您不想在这个位置恢复备份，而是想在使用其他配置的其他地方恢复备份，那么又会如何呢？同样，这种情况下恢复备份会导致问题。

重定向恢复可以解决以上问题。重定向恢复需如下 4 个步骤恢复备份：

(1) 获取输入备份副本的容器和表空间信息，可以通过在 RESTORE 命令中包括 REDIRECT 关键字来完成的。例如：

```
RESTORE DATABASE SAMPLE FROM C:\DBBACKUP
    INTO NEWDB REDIRECT WITHOUT ROLLING FORWARD
```

下面是这个命令的输出：

```
SQL1277W  A redirected restore operation is being performed.  Table space
configuration can now be viewed and table spaces that do not use automatic
storage can have their containers reconfigured.
```

(2) 从被(部分地)恢复的数据库 newdb 中查看表空间信息：

```
LIST TABLESPACES SHOW DETAIL
```

(3) 为每个表空间设置新的容器，方法如下：

```
SET TABLESPACE CONTAINERS FOR 0 USING (FILE "d:\newdb\cat0.dat" 5000)
SET TABLESPACE CONTAINERS FOR 1 USING (FILE "d:\newdb\cat1.dat" 5000)
...
SET TABLESPACE CONTAINERS FOR n USING (PATH "d:\newdb2")
```

其中 n 表示备份中某个表空间的 ID，可以从 LIST TABLESPACES 命令的输出中获得。注意，在重定向恢复中，不能改变表空间的类型。例如表空间是 SMS，不能将它变为 DMS。

(4) 通过包括关键字 CONTINUE，开始将数据本身恢复到新的容器中，如下所示：

```
RESTORE DATABASE SAMPLE CONTINUE
```

您已经看到了重定向恢复是如何工作的。它还可用于为 SMS 表空间添加容器。我们在“第 3 章：创建数据库和表空间”中讲过，在大多数情况下，不能修改 SMS 表空间并为之添加一个容器。但是，重定向恢复可以绕过这个限制。

您不需要手动执行以上 4 个步骤，重定向实用程序允许通过带 REDIRECT 和 GENERATE SCRIPT 选项发出 RESTORE 来生成一个重定向恢复脚本。当使用 GENERATE SCRIPT 选项时，恢复实用程序从备份镜像中提取容器信息，并生成一个 CLP 脚本，其中包括所有详细的容器信息。之后，您可以在这个脚本中修改任何路径或容器大小，还可以运行 CLP 脚本用一组新的容器重新创建数据库。

例如，要使用一个脚本来执行 SAMPLE 数据库的重定向恢复，可以遵循以下步骤：

(1) 使用恢复实用程序生成一个重定向恢复脚本：

```
    RESTORE DATABASE SAMPLE FROM C:\DBBACKUP INTO NEWDB REDIRECT GENERATE SCRIPT
NEWDB.CLP WITHOUT ROLLING FORWARD
```

这样会创建一个名为 *target-database-alias*.clp 的重定向恢复脚本。在这个例子中就是 newdb.clp。

(2) newdb.clp 包含执行重定向恢复所需的所有命令。它还包含所有表空间信息。下面显示了 newdb.clp 脚本。

```
*********************************************************************
-- ** automatically created redirect restore script
*********************************************************************
UPDATE COMMAND OPTIONS USING S ON Z ON NEWDB NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM  0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
*********************************************************************
-- ** automatically created redirect restore script
```

```
*********************************************************************
RESTORE DATABASE SAMPLE
-- USER  username
-- USING 'password'
FROM 'C:\dbbackup' TAKEN AT 20080516120102
-- ON 'D:'
-- DBPATH ON 'target-directory'
INTO NEWDB
-- NEWLOGPATH 'D:\DB2\NODE0000\SQL00001\SQLOGDIR\'
-- WITH num-buff BUFFERS
-- BUFFER buffer-size
-- REPLACE HISTORY FILE
-- REPLACE EXISTING REDIRECT
-- PARALLELISM n WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING;
*********************************************************************
-- ** table space definition
*********************************************************************
-- ** Tablespace name                              = SYSCATSPACE
-- **   Tablespace ID                              = 0
-- **   Tablespace Type                            = Database managed space
-- **   Tablespace Content Type                    = All permanent data. Regular
table space.
-- **   Tablespace Page size (bytes)               = 4096
-- **   Tablespace Extent size (pages)             = 4
-- **   Using automatic storage                    = Yes
-- **   Auto-resize enabled                        = Yes
-- **   Total number of pages                      = 16384
-- **   Number of usable pages                     = 16380
-- **   High water mark (pages)                    = 8872
*********************************************************************
-- ** Tablespace name                               = TEMPSPACE1
-- **   Tablespace ID                               = 1
-- **   Tablespace Type                             = System managed space
-- **   Tablespace Content Type                     = System Temporary data
-- **   Tablespace Page size (bytes)                = 4096
-- **   Tablespace Extent size (pages)              = 32
-- **   Using automatic storage                     = Yes
-- **   Total number of pages                       = 1
*********************************************************************
-- ** Tablespace name                               = USERSPACE1
-- **   Tablespace ID                               = 2
-- **   Tablespace Type                             = Database managed space
```

```
    -- **  Tablespace Content Type              = All permanent data. Large
table space.
    -- **  Tablespace Page size (bytes)         = 4096
    -- **  Tablespace Extent size (pages)       = 32
    -- **  Using automatic storage              = Yes
    -- **  Auto-resize enabled                  = Yes
    -- **  Total number of pages                = 8192
    -- **  Number of usable pages               = 8160
    -- **  High water mark (pages)              = 1888
    ********************************************************************
    -- ** Tablespace name                       = TBS1
    -- **  Tablespace ID                        = 5
    -- **  Tablespace Type                      = Database managed space
    -- **  Tablespace Content Type      = All permanent data. Large table  space
    -- **  Tablespace Page size (bytes)         = 4096
    -- **  Tablespace Extent size (pages)       = 32
    -- **  Using automatic storage              = No
    -- **  Auto-resize enabled                  = No
    -- **  Total number of pages                = 1000
    -- **  Number of usable pages               = 960
    -- **  High water mark (pages)              = 96
    ********************************************************************
    SET TABLESPACE CONTAINERS FOR 5
    -- IGNORE ROLLFORWARD CONTAINER OPERATIONS
    USING (FILE  'd:\DB2\SAMPLE\tbs1'  1000);
    ********************************************************************
    -- ** start redirected restore
    ********************************************************************
    RESTORE DATABASE NEWDB CONTINUE;
    ********************************************************************
    -- ** end of file
    ********************************************************************
```

'--'表明是注释。SET TABLESPACE CONTAINER 命令只是用于没有设置为使用自动存储的表空间。对于使用自动存储的表空间，它们的容器是由 DB2 自动处理的，因此不需要重新设置它们。

(3) 通过在文本编辑器中打开重定向恢复脚本，可以对其进行修改。您可以修改：

- 恢复选项
- 容器布局和路径

(4) 运行修改后的重定向恢复脚本：

```
db2 -tvf newdb.clp
```

这个脚本的输出将被写到一个名为 *databasename_nodenumber*.out 的文件中。

7.4.6　恢复已 drop 的表

您可能会偶然 drop 包含仍需要的数据的表。如果是这样，您应该考虑在 drop 表操作后使关键的表成为可复原的。可通过调用数据库复原操作恢复表数据，后跟一个前滚到 drop 表前的某时间点的数据库前滚操作。如果数据库很大，那么可能会花很多时间，并使数据在恢复期间不可用。已 drop 的表的恢复功能使您可以使用表空间级的复原和前滚操作来恢复已 drop 的表数据。这样会比数据库级的恢复要快，而且您的数据库将对用户保持可用。

要使已 drop 的表可以复原，必须对该表所在的表空间启用 DROPPED TABLE RECOVERY 选项。这可以在表空间创建期间或通过调用 ALTER TABLESPACE 语句来完成，这个选项在 DB2 V9 中默认是启用的。DROPPED TABLE RECOVERY 选项是表空间特定的，并限于对常规表空间使用。要确定是否对表空间启用了已 drop 的表的恢复功能，可以查询 SYSCAT.TABLESPACES 目录表中的 DROP_RECOVERY 列。

创建表空间时，默认情况下已 drop 的表的恢复选项已打开。如果不想启用表空间的已 drop 的表的恢复功能，那么可以在发出 CREATE TABLESPACE 命令时显式将 DROPPED TABLE RECOVERY 选项设置为 OFF，或者可以使用 ALTER TABLESPACE 命令来对现有表空间禁用已 drop 的表的恢复功能。如果有许多 drop 表操作要恢复或者历史记录文件很大，那么正向恢复时已 drop 的表的恢复功能可能会影响性能。

对表(对该表的表空间启用了已 drop 的表的恢复功能)运行 DROP TABLE 语句时，将在日志文件中建立另一条目(标识已 drop 的表)。还会在恢复历史记录文件中建立一个条目，包含可用于重新创建表的信息。

对可从已 drop 的表中复原的数据的类型有一些限制。不可能复原：

- 大对象(LOB)或长字段数据。对于大型表空间，不支持 DROPPED TABLE RECOVERY 选项。如果尝试复原包含 LOB 或 LONG VARCHAR 列的已 drop 的表，这些列将在生成的导出文件中设置为 NULL。仅可对常规表空间使用 DROPPED TABLE RECOVERY 选项，而不能对临时或大型表空间使用。
- XML 数据。如果尝试恢复包含 XML 数据的已 drop 的表，那么相应的列数据将为空。

如果在 drop 表时它处于重组暂挂状态，那么历史记录文件中的 CREATE TABLE DDL 与导入文件的 CREATE TABLE DDL 不完全匹配。在执行重组建议的第一个 ALTER 操作之前，导入文件将采用该表的格式，但历史记录文件中的 CREATE TABLE 语句将与包含任何 ALTER TABLE 语句结果的表的状态相匹配。

如果正在恢复 GRAPHIC 或 VARGRAPHIC 数据类型的数据，那么该数据可能会包括

多个代码页。为恢复此数据，需要指定 IMPORT 或 LOAD 命令的 USEGRAPHICCODEPAGE 文件类型修饰符。在这种情况下，使用 LOAD 命令来恢复数据将会提高恢复操作的性能。

一次只能复原一个已 drop 的表。可执行下列操作来复原已 drop 的表：

(1) 通过调用 LIST HISTORY DROPPED TABLE 命令来标识已 drop 的表。已 drop 的表标识列示在“备份标识”列中。

(2) 复原在 drop 该表前所建立的数据库级或表空间级备份镜像。

(3) 创建包含表数据的文件将写至的导出目录。此目录必须可供所有数据库分区访问，或者在每个数据库分区上都存在。此导出目录下的子目录是由每个数据库分区自动创建的。这些子目录的名称是 NODEnnnn，其中 nnnn 代表数据库分区或节点号。包含已 drop 的表数据的数据文件(就如它存在于每个数据库分区上那样)将导出到称为 data 的较低子目录中，例如\export_directory\NODE0000\data

(4) drop 表后前滚至某时间点，对 ROLLFORWARD DATABASE 命令使用 RECOVER DROPPED TABLE 选项。也可前滚至日志末尾，以使对表空间或数据库中的其他表进行的更新不会丢失。

(5) 使用 CREATE TABLE 语句从恢复历史记录文件重新创建表。

(6) 将在前滚操作期间导出的表数据导入表中。如果进行 drop 时表处于重组暂挂状态，那么需要更改 CREATE TABLE DDL 的内容以与数据文件的内容相匹配。

下面我们举一个 dropped table recovery 的例子。

(1) 执行完全数据库备份，需要注意备份镜像的时间戳。

(2) 连接到数据库并创建表，执行生成日志记录的操作，插入几条记录：

```
CONNECT TO sample
   CREATE TABLE tab1(no INTEGER) IN ts1
   INSERT INTO tab1 VALUES(1),(2),(3),(4),(5)
   ARCHIVE LOG FOR DATABASE sample
       SELECT * FROM tab1 /* check the 5 committed values from TAB */
```

(3) 模拟意外丢弃表的场景：

```
DROP TABLE tab1
   COMMIT
   SELECT * FROM tab1
```

将返回以下错误消息：

```
Error: SQL0204N "Administrator.TAB1" is an undefined name
```

(4) 恢复数据库。

要恢复已被丢弃的表，先恢复数据库备份，然后执行向前恢复(rollforward)操作：

```
RESTORE DATABASE sample FROM c:\backup TAKEN AT 20080314144204 INTO sample
```

将返回以下消息：

```
SQL2539W Warning! Restoring to an existing database that is the same as the
Backup image database.
    The database files will be deleted.
        Do you want to continue? (Y/N)      按 Y 键完成此过程。
```

(5) 检索已丢弃表的对象 ID。

使用以下命令检索意外丢弃的表的对象 ID：

```
LIST HISTORY DROPPED TABLE ALL FOR DATABASE sample
```

可以将返回的信息(例如表 7-3 中显示的示例)复制到某个文本文件中以供未来引用。

表 7-3　LIST HISTORY 命令返回的信息

```
Op Obi Timestamp Sequence Type Dev Earliest Log Current Log  Backup ID
-- --- ------------------ ---- --- ------------ ------------ ----------------------------
D  T  20080314142913                  000000000000892700050108
--------------------------------------------------------------------------------------------
"ADMINISTRATOR"."TAB1" resides in 1 table space(s):
00001 TS1
--------------------------------------------------------------------------------
Comment: DROP TABLE
Start Time: 20080314142913
End Time: 20080314142913
Status: A
--------------------------------------------------------------------------------
EID: 37
DDL: CREATE TABLE "ADMINISTRATOR"."TAB1" ( "NO" INTEGER )  IN "TS1" ;
```

表 7-3 中的 Backup ID 栏显示被丢弃的表的 ID 为 000000000000892700050108。这一信息对于恢复表非常重要。

(6) 向前恢复数据库。

现在已经获得了被丢弃的表的 ID，下一步需要使用该表的备份 ID 恢复数据库，这样

才能够导入表的数据。在向前恢复数据库之前，需要确保有一个目录可供存储导入数据，比如说 c:\sample\exporttab1。使用以下命令向前恢复数据库：

```
    ROLLFORWARD DATABASE sample TO END OF LOGS    AND STOP RECOVER DROPPED TABLE
000000000000892700050108 TO c:\sample\exporttab1
```

END OF LOGS 选项的作用是让 DB2 在执行备份操作后应用所有可用日记文件。

(7) 检查导入的数据文件。

完成数据库向前恢复之后，需要检查在 ROLLFORWARD 命令中指定的路径。应该能够找到一个. TXT 文件，打开该文件并验证其中包含的数据与意外丢弃表之前的数据是否相同。

(8) 连接到数据库并重新创建被丢弃的表。

验证导出文件之后，我们需要重新创建被丢弃的表并重新填入数据。被丢弃的表的定义包含在步骤(5)的 LIST HISTORY 命令的输出中。连接到数据库并执行 CREATE TABLE 语句：

```
    CONNECT TO sample
       CREATE TABLE "ADMINISTRATOR"."TAB1" ( "NO" INTEGER ) IN "TS1"
```

(9) 导入数据。

重新创建表之后，可以使用以下命令将数据库重新导入到表中：

```
    IMPORT FROM c:\sample\exporttab1\Node0000\data.txt OF DEL INSERT INTO
administrator.tab1
```

IMPORT 工具将导出文件中的所有数据导回到表中，并在成功后发送报告(未显示)。

(10) 验证恢复后的数据

确保 IMPORT 过程中没有错误或报警，并且所有数据都已导回表中：

```
    SELECT * FROM tab1
```

如果一切运行正常，则意外丢弃点之前的所有数据应该都在表中。

7.5 数据库和表空间前滚

7.5.1 数据库前滚

ROLLFORWARD 命令允许时间点恢复，即该命令允许遍历 DB2 日志，将日志中记录的操作重新执行或撤销到一个指定的时间点。虽然可以将数据库或表空间前滚到最小时间

点之后的任何时间点，但是不能保证前滚到那个时间后所有数据都处于一致状态。

虽然本节不讨论 QUIESCE 命令，但是值得一提的是，可以使用这个命令在常规数据库操作期间设置一致点。通过设置这些一致点，总可以使时间点恢复达到任何一个一致点，从而保证数据同步。

一致点和很多其他信息一起存储在 DB2 历史文件中，这个文件可以使用 LIST HISTORY 命令来查看。

在前滚处理期间，DB2 将：

- 在当前日志路径中查找所需的日志文件。
- 如果找到了日志文件，则应用该日志文件中的事务。
- 如果在当前日志路径中没有发现那个日志文件，则搜索 OVERFLOW LOG PATH 选项指定的路径，并使用那个位置的日志文件。
- 如果在当前路径中没有发现那个日志文件，并且没有使用 OVERFLOW LOG PATH 选项，则使用 LOGARCHMETH1 中指定的方法从归档路径中获取日志文件。
- 如果该日志在当前日志路径或 OVERFLOW LOG PATH 中，则重新应用(apply)事务。

要执行 ROLLFORWARD 命令，需要 SYSADM、SYSCTRL 或 SYSMAINT 权限。

ROLLFORWARD 命令的语法如下：

```
ROLLFORWARD DATABASE database-alias [USER username [USING password]]
[TO {isotime [ON ALL DBPARTITIONNUMS] [USING LOCAL TIME | USING UTC TIME] |
END OF LOGS [On-DbPartitionNum-Clause]}] [AND {COMPLETE | STOP}] |
{COMPLETE | STOP | CANCEL | QUERY STATUS [USING LOCAL TIME | USING UTC TIME]}
[On-DbPartitionNum-Clause] [TABLESPACE ONLINE | TABLESPACE (tblspace-name
[ {,tblspace-name} ... ]) [ONLINE]] [OVERFLOW LOG PATH (log-directory
[{,log-directory ON DBPARTITIONNUM db-partition-number} ... ])] [NORETRIEVE]
[RECOVER DROPPED TABLE dropped-table-id TO export-directory]

On-DbPartitionNum-Clause:
  ON {{DBPARTITIONNUM | DBPARTITIONNUMS} (db-partition-number
  [TO  db-partition-number] , ... ) | ALL DBPARTITIONNUMS [EXCEPT
  {DBPARTITIONNUM | DBPARTITIONNUMS} (db-partition-number
  [TO db-partition-number] , ...)]}
```

我们来看一个例子。要执行 sample 数据库的前滚，可以使用以下任何语句：

```
(1)ROLLFORWARD DATABASE sample TO END OF LOGS AND COMPLETE
(2)ROLLFORWARD DATABASE sample TO timestamp AND COMPLETE
(3)ROLLFORWARD DATABASE sample TO timestamp USING LOCAL TIME AND COMPLETE
```

从上面的代码中：

(1) 在这个例子中，我们将前滚到日志的最后，这意味着所有归档的日志和活动日志都将被遍历。最后，它将完成前滚，并通过回滚任何未提交的事务来脱离 rollforward-pending 状态。

(2) 对于这个例子，DB2 将前滚到指定的时间点。使用的时间戳必须是 CUT(格林威治时间)，这个时间可以通过将当前时区减去当地时间计算出来。

(3) 这个例子类似于前一个例子，但是时间戳可以用当地时间表示。

语法中没有关键字 OFFLINE，因为这是默认模式。对于 ROLLFORWARD 命令，这是可用于数据库的唯一模式。

7.5.2 表空间前滚

表空间前滚通常可以在线进行或者离线进行。一个例外就是系统编目表空间(SYSCATSPACE)，这种表空间只能离线前滚。

下面是表空间前滚的一个示例：

```
ROLLFORWARD DATABASE sample  TO END OF LOGS AND COMPLETE
  TABLESPACE ( userspace1 ) ONLINE
```

在 7.5.1 小节中我们解释了这个例子中的选项。这里唯一没有解释过的是 TABLESPACE 选项，该选项指定要前滚的表空间。

1. 关于表空间前滚的考虑

如果启用了注册表变量 DB2_COLLECT_TS_REC_INFO，那么只需处理恢复表空间所需的日志文件。ROLLFORWARD 命令将忽略不需要的日志文件，这样可以缩短恢复时间。

ROLLFORWARD 命令的 QUERY STATUS 选项可用于列出：

- DB2 已经前滚的日志文件
- 需要的下一个归档日志文件
- 自前滚处理开始以来最近提交的事务的时间戳

例如：

```
ROLLFORWARD DATABASE sample QUERY STATUS USING LOCAL TIME
```

在一个表空间时间点前滚操作完成之后，表空间被置于 backup-pending 状态。这里必须使用表空间或数据库的备份，因为在表空间所恢复到的时间点与当前时间之间对表空间的更改已经丢失。

2. 使用控制中心执行前滚操作

除了可以使用命令行进行前滚数据库以外，我们还可以通过控制中心执行前滚操作。图 7-15 展示了如何从控制中心调用 ROLLFORWARD 命令。要执行一个数据库前滚或表空间前滚，可以在要前滚的数据库上单击右键并选择“**前滚**”。

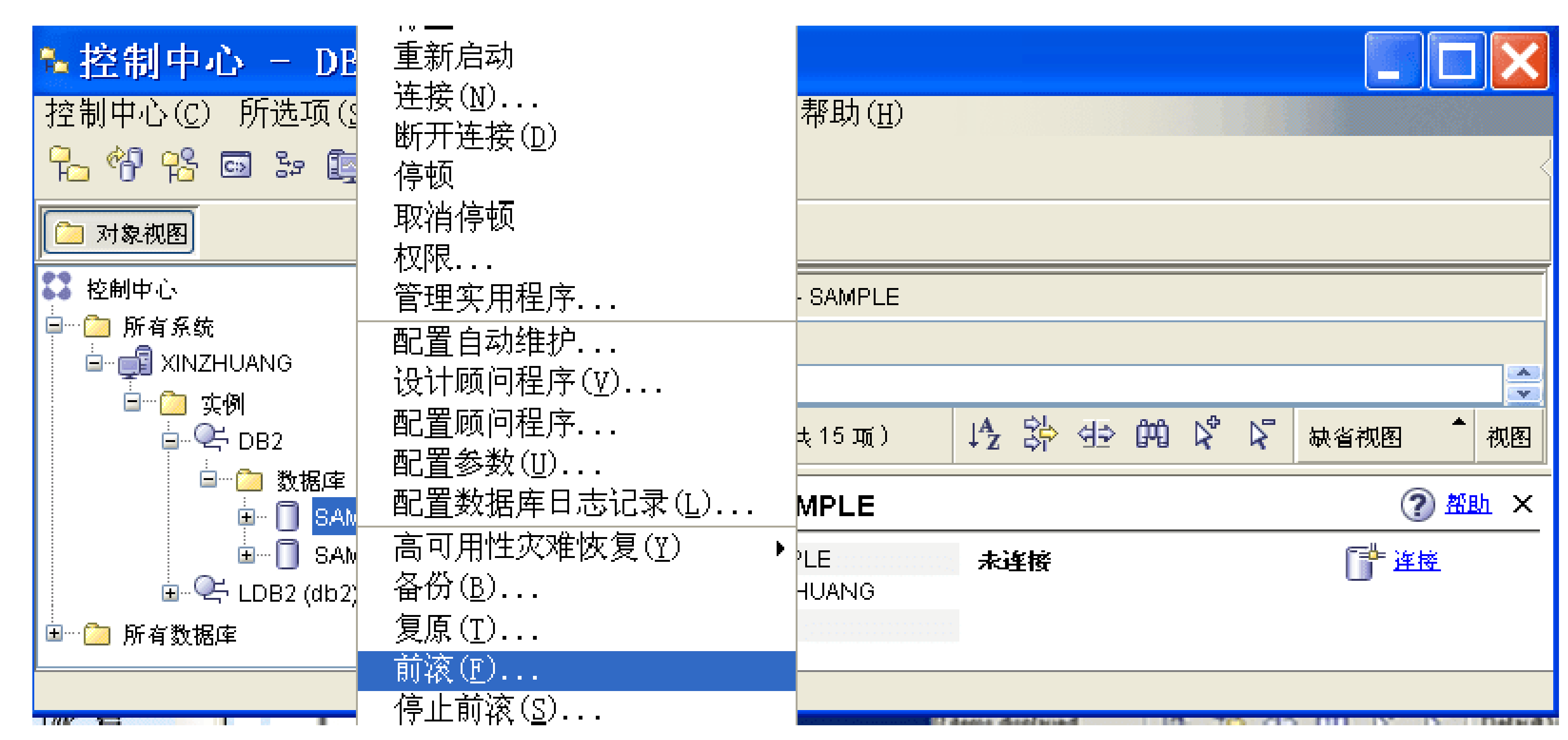


图 7-15　从控制中心调用 ROLLFORWARD 命令

图 7-16 展示了执行 ROLLFORWARD 命令所需的一些选项。

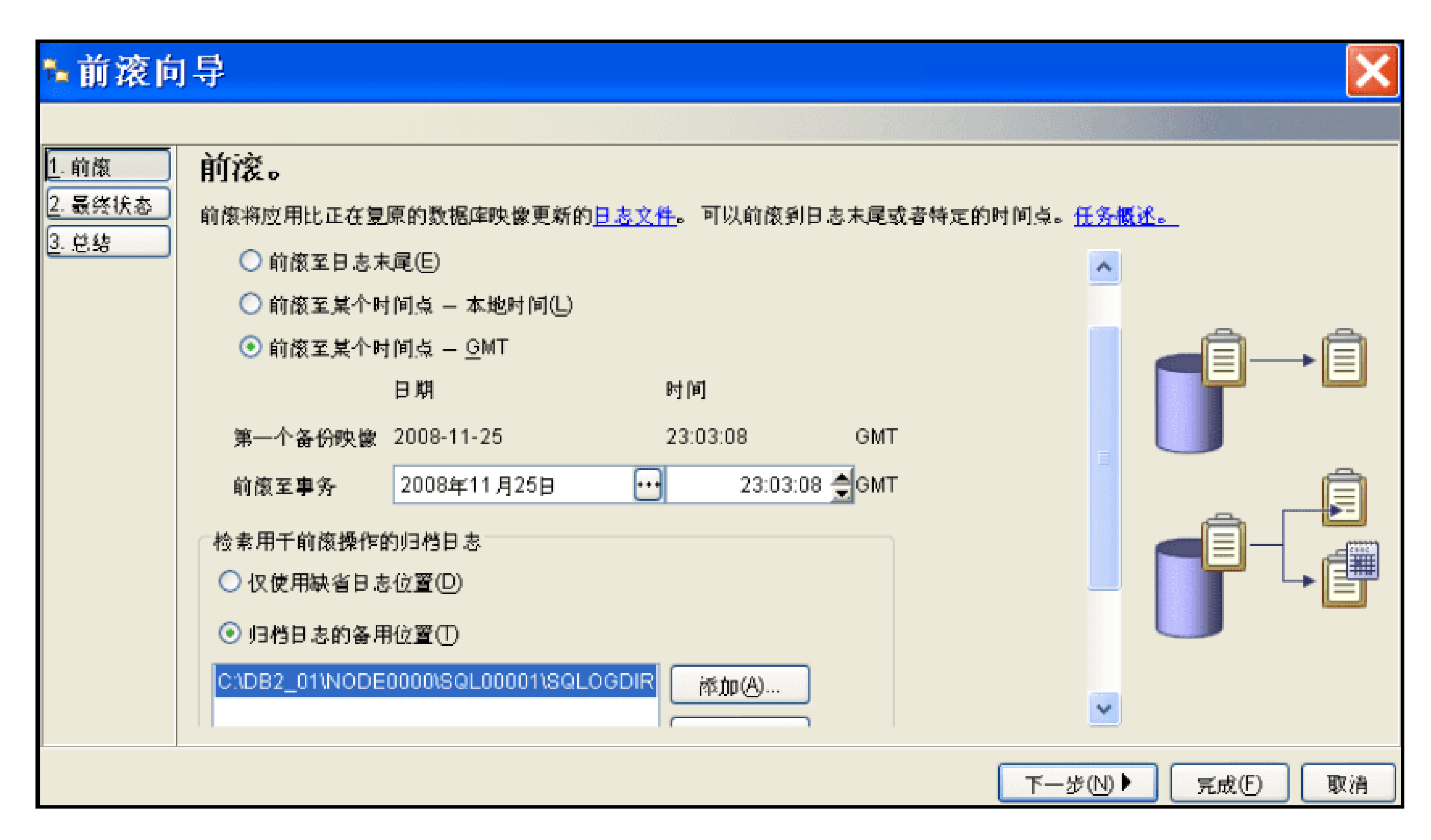


图 7-16　执行 ROLLFORWARD 命令所需的选项

3. 恢复示例

到目前为止，我们讲解了 BACKUP、RESTORE 和 ROLLFORWARD 命令。接下来我们举了一些场景来帮助我们理解不同类型的恢复。

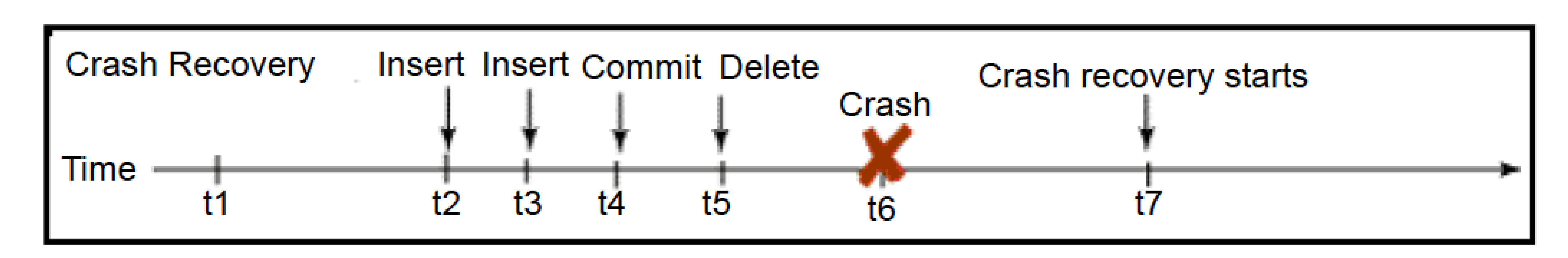


图 7-17　场景 1

对于图 7-17 所示的这个场景，使用循环日志：

在 t6，在构建中出现了一个计划外的停电事故。在 t7，DB2 被重新启动，当连接到数据库时，紧急事故恢复自动启动(假设 db cfg AUTORESTART 为 ON；否则，需要用 RESTART DATABASE 命令手动地启动它)。紧急事故恢复将遍历活动日志，并重新执行提交的事务。如果一个事务还没有被提交，它将回滚(撤销)。对于这个例子，两个插入将重新执行，而删除语句将撤销。

对于图 7-18 所示的这个场景，循环日志记录在生效：

在 t7，您意识到所有表空间中的数据已经被 t6 时启动的某个事务毁坏。在 t8，您决定使用 t1 时做的完整数据库备份进行恢复。由于循环日志记录在生效，日志中很多已提交和写到硬盘上的事务已经被覆盖。因此，不能应用日志(不能在循环日志记录中运行 ROLLFORWARD 命令，所以甚至不能前滚活动日志)。结果是：t2 至 t4 这段时间内很多好的事务将丢失。

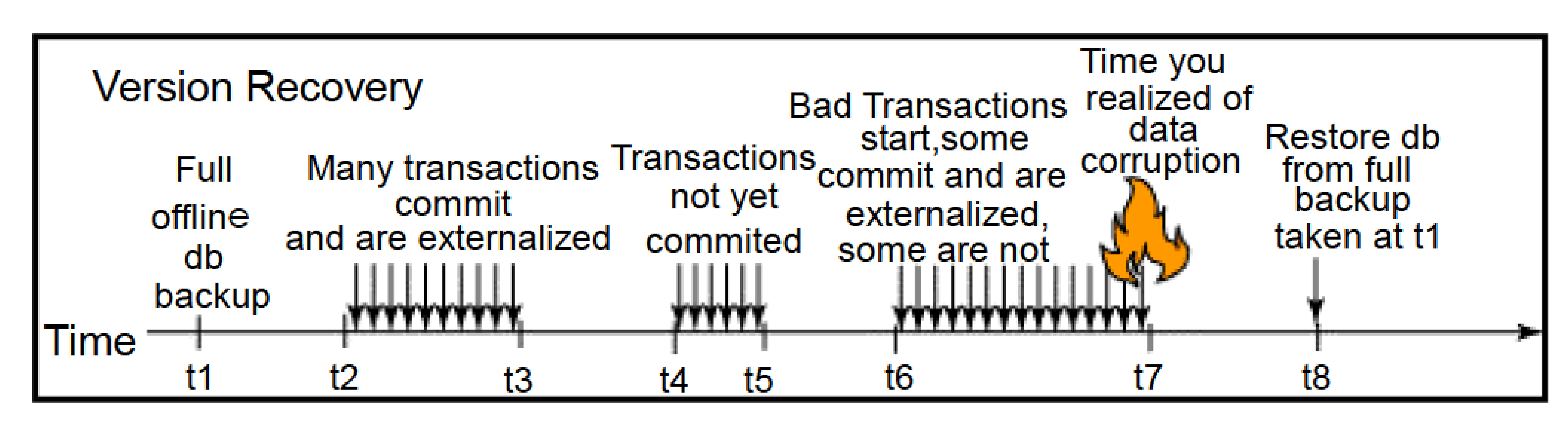


图 7-18　场景 2

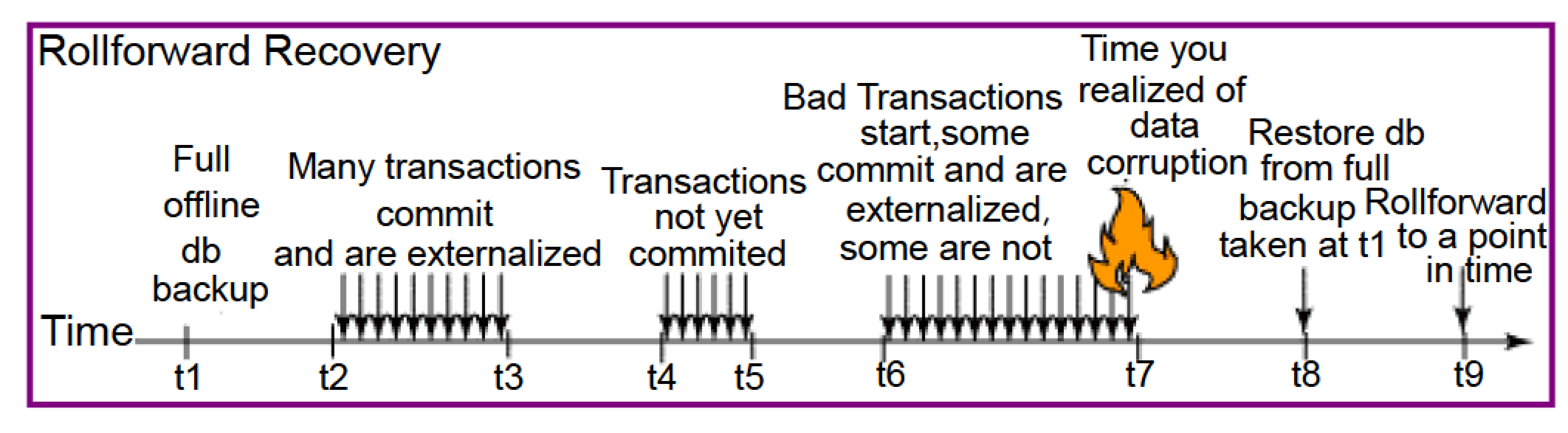


图 7-19　场景 3

对于图 7-19 所示的这个场景，使用归档日志：

这是场景 2 的一个扩展。在这个例子中，日志已经被保存(归档日志)；在 t8 应用完整数据库恢复之后，可以将日志前滚到 t9。日志可以从 t1 前滚到任意时间点，但是您很可能不想超过 t6，因为在 t6 开始了坏的事务。

图 7-20 所示的场景更详细地回顾了所有这些概念。

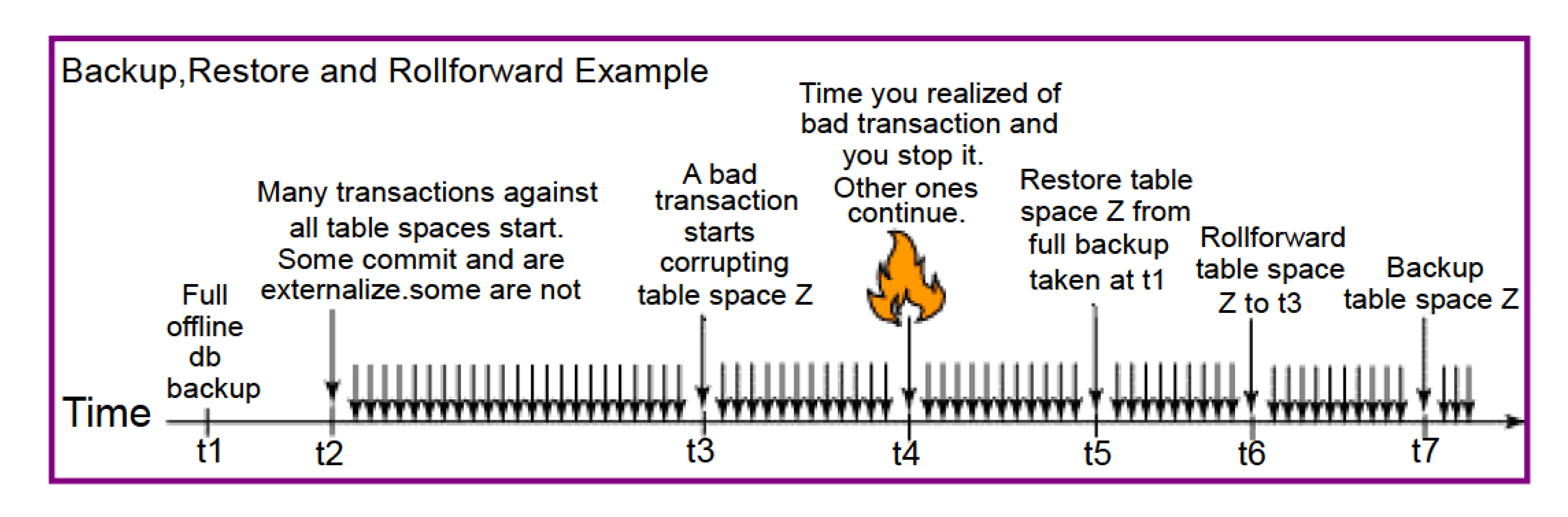


图 7-20　场景 4

t1 时完成了一次离线数据库备份。

t2 时执行日常事务。

在 t4，您意识到某个事务已经毁坏了表空间 Z，并且停止这个事务。针对其他表空间的其他事务则继续。

只对于表空间 Z，您希望数据库处于 t3 之前的状态(开始坏事务之前的状态)，因此：

- 在 t5，您使用 t1 做的完整离线备份恢复 Z。
- 完成恢复后，表空间将处于 rollforward pending 状态。
- 在 t6，您将这个表空间前滚到 t3，即坏事务开始之前。
- 您已经执行了一个时间点恢复。因此，DB2 现在将由于一致性的原因使表空间处于 backup pending 状态。
- 在 t7，您备份表空间。这时，数据库是一致的，所有用户和应用程序可以正常工作。被恢复的表空间在 t3 到 t7 之间将存在一个间隔，这就是我们的意图——删除被毁坏的数据。

7.6　RECOVER 实用程序

DB2 从 V8.2 开始提供一个新命令 RECOVER DATABASE，新的 RECOVER DATABASE 命令结合了 RESTORE DATABASE 和 ROLLFORWARD DATABASE 命令的功能。过去我

们在恢复数据库时先使用 RESTORE 命令把数据库恢复到备份的那个时刻，然后再使用 ROLLFORWARD 前滚(redo)日志中的 SQL 操作。那么能不能把这两个命令结合到一起呢？这就是 RECOVER 实用程序的由来，简单来说，recover=restore+rollforward。所以你如果仍然习惯过去那种传统的恢复方式(restore+rollforward)，则可以不用这个实用程序。使用此命令时，根据恢复历史文件中的信息使数据库恢复到一个指定的时间。它自动选择最佳适用备份镜像来执行恢复操作。要指定想要数据库恢复到的时间点，您不需要指示必须复原哪个数据库备份镜像或需要哪些日志文件以达到指定的时间点。RECOVER DATABASE 命令还支持恢复到日志文件末尾的操作。

RECOVER DATABASE 命令的语法是：

```
RECOVER DATABASE source-database-alias TO isotime [USING LOCAL TIME]
[USER username [USING password]
[USING HISTORY FILE history-file]
[OVERFLOW LOG PATH directory]
[RESTART]
```

例 7-1 开发服务器上有一个 SAMPLE 数据库。昨夜的一次停电事故导致数据库中的数据遭到毁坏。您需要尽快恢复数据库。首先可以找到适当的备份，然后找到所需的 DB2 日志，以便前滚到停电事故之前的时间点。恢复 SAMPLE 数据库的一种更容易的方法是发出 RECOVER DB 命令，如下所示：

```
RECOVER DB sample
RECOVER DB sample TO 2008-05-21-13.50.00 USING LOCAL TIME
```

在第 1 行，DB2 从可用的最近备份镜像恢复 SAMPLE 数据库，并且将前滚到日志的最后。在第 2 行，DB2 将 SAMPLE 数据库恢复到时间点 2008-05-21-13.50.00，这是按当地时间指定的。

还记得吗，RECOVER DATABASE 实用程序依赖于数据库恢复历史文件来发现最适合用于恢复的备份镜像。在上面的恢复命令中，我们没有指定历史文件的位置。由于 SAMPLE 数据库已经在服务器上，因而 DB2 可以在数据库目录路径下找到历史文件。

如果要恢复的数据库不存在，那么必须指定历史文件的位置。

```
RECOVER DB sample TO END OF LOGS USING HISTORY FILE
 (/home/user/oldfiles/db2rhist.asc)
```

在用于存放恢复后的数据库的服务器上必须有一个有效的恢复历史文件，其中包含所需的备份镜像和日志。在上面的命令中，如果没有一个可用的恢复历史文件(由文件传输或者历史文件备份得到)，那么在运行 RECOVER DATABASE 命令之前，就必须设法从备份

镜像中提取出历史文件。在这种情况下，连续运行标准的 RESTORE 和 ROLL FORWARD 命令来恢复数据库也许更容易一些。

例 7-2　例如我们有数据库 SAMPLE 的几个备份：

```
C:\>db2 list history backup all for db sample
列示 sample 的历史文件
匹配的文件条目数= 4
Op 对象时间戳记+序列 类型 设备 最早日志 当前日志备份标识
-- --- ------------------ ---- --- ------------ ------------ --------------
B D 20081025222808001 F D S0000000.LOG S0000000.LOG
----------------------------------------------------------------------------
包含 2 表空间：
00001 SYSCATSPACE
00002 USERSPACE1
----------------------------------------------------------------------------
注释：DB2 BACKUP SAMPLE OFFLINE
开始时间：20081025222808
结束时间：20081025222829
状态：A
----------------------------------------------------------------------------
EID：1 位置：C:\Temp\SAMPLE.0\DB2\NODE0000\CATN0000\20081025
Op 对象时间戳记+序列 类型 设备 最早日志 当前日志备份标识
-- --- ------------------ ---- --- ------------ ------------ --------------
B D 20081025223024001 N D S0000001.LOG S0000001.LOG
----------------------------------------------------------------------------
包含 2 表空间：
00001 SYSCATSPACE
00002 USERSPACE1
----------------------------------------------------------------------------
注释：DB2 BACKUP SAMPLE ONLINE
开始时间：20081025223024
结束时间：20081025223045
状态：A
----------------------------------------------------------------------------
EID：2 位置：C:\Temp\SAMPLE.0\DB2\NODE0000\CATN0000\20081025
Op 对象时间戳记+序列 类型 设备 最早日志 当前日志备份标识
-- --- ------------------ ---- --- ------------ ------------ --------------
B D 20081025223239001 O D S0000003.LOG S0000003.LOG
----------------------------------------------------------------------------
包含 2 表空间：
00001 SYSCATSPACE
00002 USERSPACE1
----------------------------------------------------------------------------
注释：DB2 BACKUP SAMPLE ONLINE
```

```
开始时间：20081025223239
结束时间：20081025223300
状态：A
---------------------------------------------------------------------------
EID：3 位置：C:\Temp\SAMPLE.0\DB2\NODE0000\CATN0000\20081025
Op 对象时间戳记+序列 类型 设备 最早日志 当前日志备份标识
-- --- ------------------ ---- --- ------------ ------------ --------------
B D 20081025223408001 D D S0000005.LOG S0000005.LOG
---------------------------------------------------------------------------
包含 2 表空间：
00001 SYSCATSPACE
00002 USERSPACE1
---------------------------------------------------------------------------
注释：DB2 BACKUP SAMPLE OFFLINE
开始时间：20081025223408
结束时间：20081025223428
状态：A
---------------------------------------------------------------------------
EID：4 位置：C:\Temp\SAMPLE.0\DB2\NODE0000\CATN0000\20081025
```

我们想把数据库恢复到时间点 2008 年 10 月 25 日 22 点 30 分，只需要运行如下命令就可以了：

```
C:\>db2 recover database sample to 2008-10-25-22.30.00.000000
前滚状态
输入数据库别名 = sample
节点数已返回状态 = 1
节点号 = 0
前滚状态 = 未暂挂
下一个要读取的日志文件=
已处理的日志文件= S0000000.LOG - S0000000.LOG
上次落实的事务= 2008-10-25-22.30.00.000000
DB20000I RECOVER DATABASE 命令成功完成。
```

另外，您还可以直接恢复数据库到日志末尾，使用如下命令：

```
C:\>db2 recover database sample to end of logs
前滚状态
输入数据库别名 = sample
节点数已返回状态 = 1
节点号 = 0
前滚状态 = 未暂挂
下一个要读取的日志文件=
已处理的日志文件= S0000000.LOG - S0000001.LOG
```

```
上次落实的事务= 2008-10-25-22.30.00.000000
DB20000I RECOVER DATABASE 命令成功完成。
```

不管出于何种原因，如果一个恢复操作还没有成功完成就被中断，那么可以通过运行相同的命令来重新启动这个操作。如果是在前滚阶段被中断，那么恢复实用程序将尝试继续之前的前滚操作，而不会重复恢复阶段。如果要强制恢复实用程序重复恢复阶段，可以带 RESTART 选项发出 RECOVER DATABASE 命令，迫使恢复实用程序忽略之前没能完成的恢复操作。如果恢复实用程序在恢复阶段被中断，那么它将从头开始。

恢复实用程序不支持以下 RESTORE DATABASE 命令选项：

- TABLESPACE tablespace-name ——表空间恢复操作不受支持
- INCREMENTAL——增量恢复操作不受支持
- OPEN num-sessions SESSIONS——不能用 TSM 或另一种供应商产品指定 I/O 会话号
- BUFFER buffer-size—— 不能设置用于恢复操作的缓冲区的大小
- DLREPORT filename——不能指定已断开链接的报告文件的文件名
- PARALLELISM n——不能指定恢复操作的并行度
- WITHOUT PROMPTING——不能规定一个恢复操作在没有提示的情况下运行

7.7　恢复历史文件

我们在上面的复原中提到了恢复历史记录文件，恢复历史记录文件是与每个数据库一起创建的(图 7-21 所示)，且在发生下列情况时自动更新：

- 备份、复原、恢复和前滚了数据库或表空间
- 自动重建了数据库，并且复原了多个镜像
- 创建、改变、停顿、重命名、删除了表空间
- 装入(LOAD)、重组、runstats 了表
- DROP 表(启用已 drop 表恢复时)

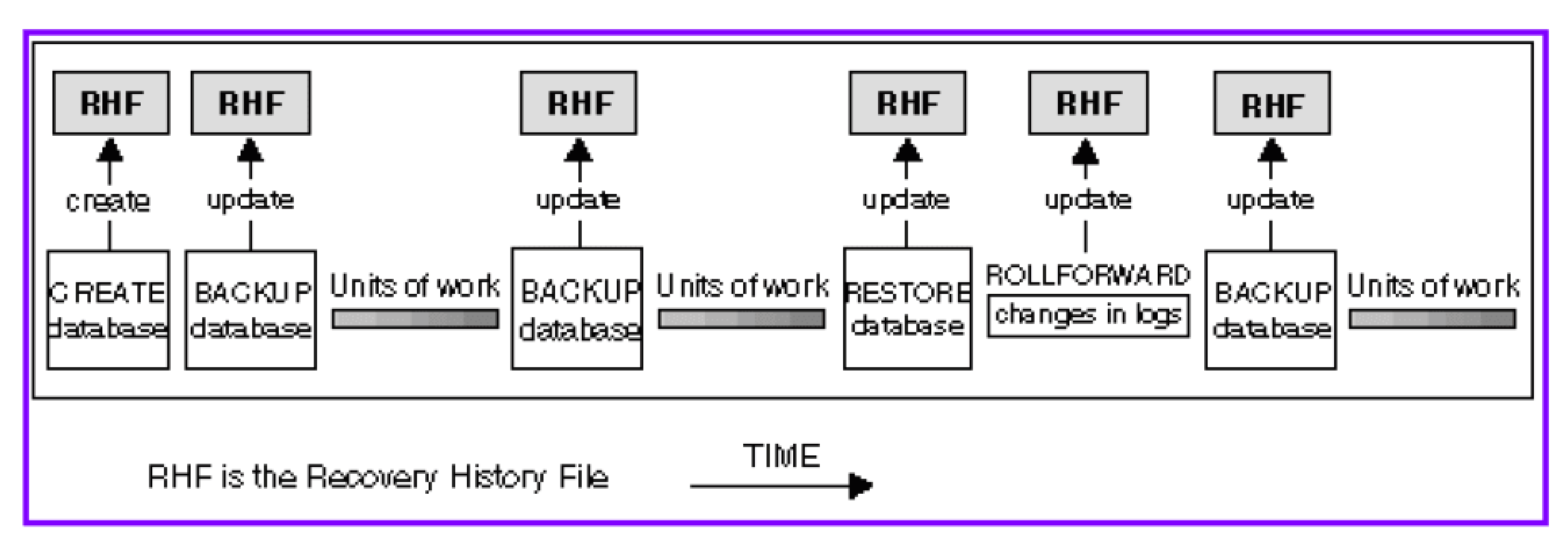


图 7-21　恢复历史记录文件

- 调用按需(archive)进行的日志归档
- 写入新日志文件(使用可恢复日志记录时)
- 归档日志文件(使用可恢复日志记录时)

可以使用恢复历史文件中汇总的备份信息，将数据库的全部或部分恢复至给定的时间点。该文件中的信息包括：

- 唯一地标识每个条目的标识(ID)字段
- 已复制的部分数据库和复制方法
- 建立副本的时间
- 副本的位置(指示设备信息和访问副本的逻辑方式)
- 上次进行复原操作的时间
- 重命名表空间的时间，显示了该表空间的先前名称和当前名称
- 备份操作的状态：活动、不活动、到期的或删除的
- 数据库备份保存的或前滚恢复操作期间处理的最后一个日志序号

要查看恢复历史记录文件中的条目，可使用 LIST HISTORY 命令。要列出对数据库 SAMPLE 完成的所有备份(BACKUP)、恢复(RESTORE)或装入(LOAD)操作，可以发出以下命令：

```
DB2 LIST HISTORY ALL FOR SAMPLE
```

每当执行备份、恢复或装入操作时，就会自动地将历史文件写入到相应的地方。用户不能把记录直接插入到历史文件中。

每个备份操作(数据库备份、表空间备份或增量备份)都包括复制恢复历史记录文件。该恢复历史记录文件与数据库相关联。删除数据库会删除恢复历史记录文件。将数据库复原至新位置会复原该恢复历史记录文件。复原不会覆盖现有恢复历史记录文件，除非磁盘上的文件没有任何条目(在这种情况下，将根据备份镜像复原数据库历史记录)。

如果由于文件系统、恢复历史文件被破坏或出现 I/O 错误，使得不能向恢复历史文件插入或更新信息，那么用户将会收到一个警告信息。如果文件系统装满，那么当前处理的数据项将被丢失。

用户可以用 PRUNE HISTORY 命令管理这一恢复历史文件。命令的语法如下：

```
PRUNE HISTORY timestamp [WITH FORCE OPTIONS]
```

时间戳等于或小于所提供时间印戳值的所有项都要从恢复历史文件中被删除。WITH FORCE OPTIONS 规定：根据所指定的时间戳文件将被修改，即使最近恢复集合中的一些项已经从该文件中被删除。

要使用 PRUNE HISTORY 命令，用户必须拥有 SYSADM、SYSCTRL、SYSMAINT 或 DBADM 权限。每个数据库都有一个恢复历史数据。这个文件的大小取决于 PRUNE HISTORY 这一数据库配置参数和备份、恢复以及表装入操作的频率。配置参数 REC_ HIS_ RETENTN 用来设置历史文件的保留期间(默认值 366 天)。如果用户想要无限期地保持恢复历史文件，可以把这一参数设置为－1。按默认方式，在记录完整个数据库备份之后，该恢复历史文件会被自动根据配置参数 REC_ HIS_ RETENTN 设置成自动修剪。

恢复历史文件的内容如表 7-4 中所示。

表 7-4　恢复历史文件的内容

列　　名	描　　述
OPERATION	执行的操作类型：B = Backup(备份)，R = Restore(恢复)，L = Load(装入)，A=Create Tablespace，N=Rename 等
OBJECT	对象标识，标识操作的对象
时间戳+序列	记录操作的时间戳和序列
OPTYPE	操作类型，例如 F=Full 数据库离线备份；N=Online，R=Replace；P = Table Space (表空间) ，T= Table (表)
DEVICE_TYPE	设备类型：D=Disk(磁盘)，K=Diskette(软盘)，T=Tape (磁带)，A=ADSM，U= UserExit (用户出口)，O =其他供应商设备
FIRST_LOG	**最早日志：对于在线备份来说，该日志表示是在线备份期间开始写入的第一个日志**
LAST_LOG	**最后日志：对于在线备份来说，表示在线备份完成后写入的最后一个日志。对于离线备份来说，最早日志和最晚日志永远相等**
BACKUP_ID	备份。前 14 个字母 = yyymmddhhmnss。后 3 个字符 = 顺序号
COMMENT	操作注释：例如备份类型
STARTIME	操作开始时间
STOPTIME	操作结束时间
LOCATION	存放位置

在恢复历史文件的输出中，最早日志和最晚日志非常重要，它们对维护离线数据库和在线数据库的完整性非常重要。最早日志：对于在线备份来说，该日志是在线备份期间开始写入的第一个日志。最后日志：对于在线备份来说，是在线备份完成后写入的最后一个日志。对于离线备份来说，最早日志和最晚日志永远相等。对于离线备份来说，我们必须保存好最晚日志之后的日志文件以执行前滚恢复。对于在线备份来说，我们必须维护好最早日志和最晚日志之间的日志文件以保证此在线备份的有效性。如果最早日志和最晚日志

之间的日志文件丢失，那么这个在线备份是无效的。这个切切注意。

如果当前数据库不能使用或者没有提供并且相关的恢复历史文件被破坏或被删除，那么 RESTORE DATABASE 命令有一个选项允许只恢复相应的恢复历史文件。这样，可以复查该恢复历史记录文件，以提供有关要用于复原该数据库的备份的信息。假如你的恢复历史文件被破坏，您可以通过下面的命令来恢复某个特定备份版本的恢复历史文件。

```
DB2 restore db sample history file  --这个操作只恢复恢复历史文件
```

7.8 数据库重建

7.8.1 数据库重建概念

数据库重建(REBUILD)功能是由恢复实用程序提供的。它允许使用一组备份镜像重建一个全新的数据库。您可以选择重建整个数据库，或者重建一个只包含原来数据库中部分表空间的数据库。数据库重建过程取决于数据库是可恢复的还是不可恢复的。在后面的小节中我们将先后谈到这两种场景。

7.8.2 使用表空间备份重建可恢复数据库

对于可恢复数据库，重建实用程序允许只使用表空间备份重建整个数据库。这里不需要完整数据库备份。完整数据库备份可能需要更大的维护窗口，对于高可用性环境，这样会增加调度的难度。使用表空间备份重建数据库的能力对于可用性和可恢复性来说是个很好的增强。

假设您有一个名为 TEST 的可恢复数据库。某天夜里，出现了一次停电事故。数据库所在的磁盘遭到了损坏。数据库再也不能访问了，于是您想恢复数据库。该数据库有以下表空间：

- SYSCATSPACE(系统编目)
- USERSPACE1(用户数据表空间)
- USERSPACE2(用户数据表空间)
- USERSPACE3(用户数据表空间)

您可以用的还有：

- 所有日志文件。由于日志与数据库存储在不同的磁盘上，因此它们能够幸免于难。
- 您没有任何数据库级的备份，但是有以下表空间备份：

```
    TEST.3.DB2.NODE0000.CATN0000.20080515135047.001 — SYSCATSPACE 和
USERSPACE1 的备份
```

```
    TEST.3.DB2.NODE0000.CATN0000.20080516135136.001 — USERSPACE2 和
USERSPACE3 的备份
    TEST.3.DB2.NODE0000.CATN0000.20080517135208.001 — USERSPACE3 的备份
```

如果我们使用恢复和前滚方法(前面的小节中有过讲解)，将数据库恢复到最近的一个时间点，那么我们需要恢复一个数据库备份，然后将数据库前滚到日志的最后。不幸的是，在这种情况下，这是不可能实现的，因为我们没有数据库备份，我们只有表空间备份。如果在任何一个表空间备份上运行一个常见的 RESTORE 命令，那么将得到以下错误：

```
    db2 restore db test taken at 20080517135208
    SQL2560N The target database is not identical to the source database for
a restore from a table space level backup.
```

有了数据库重建功能，现在可以只用表空间备份和日志重建 TEST 数据库。要重建一个数据库，可以在 RESTORE DATABASE 命令中指定 REBUILD 选项。

下面的步骤将 TEST 数据库重建到最近的时间点。

(1) 带 REBUILD 选项发出 RESTORE DATABASE 命令：

```
    restore db test rebuild with all tablespaces in database taken at 20080517135208
```

重建过程中的第一步是识别重建目标镜像。重建目标镜像应该是要在重建操作中使用的最近的备份镜像。之所以称之为目标镜像，是因为它定义了要重建的数据库的结构，包括可以恢复的表空间、数据库配置和日志序号。它可以是任何类型的备份(完整备份、表空间备份、增量备份、在线或离线备份)。在这个例子中，最近的备份镜像是 TEST.3.DB2.NODE0000.CATN0000.20080517135208.001，因此我们使用它作为重建操作的目标镜像。

在这个命令成功执行之后，TEST 数据库的结构被恢复。我们可以得到数据库配置和它的历史之类的信息。如果发出一个 LIST HISTORY 命令(例如 db2 list history all for test)，那么我们将得到以下输出：

```
    Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
    -- --- ------------------ ---- --- ------------ ------------ --------------
     R  D  20080519121107001   F                              20080517135208
    ---------------------------------------------------------------------------
     Contains 1 tablespace(s):
     00001 USERSPACE3
    ---------------------------------------------------------------------------
       Comment: RESTORE TEST WITH RF
    Start Time: 20080519121107
      End Time: 20080519121108
```

```
    Status: A
 ----------------------------------------------------------------------------
  EID: 7 Location:
 Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
 -- --- ------------------ ---- --- ------------ ------------ --------------
  R  P  20080519121108001   F                            20080515135047
 ----------------------------------------------------------------------------
  Contains 2 tablespace(s):
  00001 USERSPACE1
  00002 SYSCATSPACE
 ----------------------------------------------------------------------------
   Comment: RESTORE TEST WITH RF
 Start Time: 20080519121108
   End Time: 20080519121113
     Status: A
 ----------------------------------------------------------------------------
  EID: 8 Location:
 Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
 -- --- ------------------ ---- --- ------------ ------------ --------------
  R  P  20080519121113001   F                            20080516135136
 ----------------------------------------------------------------------------
  Contains 1 tablespace(s):
  00001 USERSPACE2
 ----------------------------------------------------------------------------
   Comment: RESTORE TEST WITH RF
 Start Time: 20080519121113
   End Time: 20080519121114
     Status: A
 ----------------------------------------------------------------------------
  EID: 9 Location:
 Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
 -- --- ------------------ ---- --- ------------ ------------ --------------
  R  D  20080519121107      R       S0000001.LOG S0000003.LOG 20080518135208
 ----------------------------------------------------------------------------
  Contains 4 tablespace(s):
  00001 USERSPACE3
  00002 USERSPACE2
  00003 USERSPACE1
  00004 SYSCATSPACE
 ----------------------------------------------------------------------------
   Comment: REBUILD TEST WITH RF
 Start Time: 20080519121107
   End Time: 20080519121115
```

```
    Status: A
  ----------------------------------------------------------------------------
 EID: 10 Location:
```

上面显示的 LIST HISTORY 命令输出中有 4 个条目，它们都与重建操作有关。

第一个条目是 EID：7，它表明是备份镜像 20080517135208 上的一个恢复操作，被恢复的表空间为 USERSPACE3(还记得吗，这个备份镜像只包含 USERSPACE3)。但是，我们需要使用 ALL TABLESPACES 选项恢复所有表空间，所以数据库中剩下的表空间也要恢复。这一点反映在 LIST HISTORY 输出中剩下的部分。

通过使用备份恢复文件中的信息，恢复实用程序发现要恢复的所有表空间的备份镜像，并恢复它们。在恢复之后，表空间被置于 rollforward pending 状态。您可以看看 LIST HISTORY 输出中的注释行，每个表空间都标上了'WITH RF'，表明在恢复之后要执行前滚。

要进行恢复，所有备份镜像都必须已经存在，并存储在历史文件中。否则就会返回一个错误，说明恢复实用程序不能发现所需的镜像。

(2) 带 TO END OF LOGS 选项发出一个 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test to end of logs
```

在恢复了所有表空间之后，这些表空间被置于 rollforward pending 状态。我们需要前滚数据库，使数据库回到正常状态。

要在重建操作期间前滚数据库，最早备份镜像与最近备份镜像之间的所有日志文件都必须能够为前滚实用程序所用。如果您想前滚到比最近备份更近的时间点上，那么这个备份之后的所有日志文件也必须可用。

在我们的例子中，所有日志仍然完好无损，它们仍然存在于 LOGPATH 数据库配置参数指定的日志路径中。前滚实用程序将在那里找到它们。正因为如此，我们不需要在 ROLLFORWARD 命令中指定日志文件的路径。如果日志文件被存储在其他地方，那么就必须使用 ROLLFORWARD 命令中的 OVERFLOW LOG PATH 选项指定日志文件的位置。

(3) 带 STOP 选项发出一个 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test stop
```

至此，TEST 数据库已经可以连接，并且所有表空间都处于 NORMAL 状态。

7.8.3　只使用部分表空间备份重建可恢复数据库

如 7.8.2 节所演示的那样，数据库重建功能让我们可以只使用表空间备份和日志来重建一个完整的数据库。这个实用程序是如此健壮，以至于我们不需要所有的表空间备份就能重建一个数据库。我们可以只用一部分表空间就能重建一个数据库。

再次使用 7.8.2 节的例子，假设 USERSPACE1 和 USERSPACE2 中的数据对于用户来说非常重要。在停电事故发生后，我们必须尽快恢复这两个表空间。USERSPACE3 则不太重要，而且它很大。如果恢复所有表空间，那么就要花很长的时间。如果可以只用其中的 USERSPACE1 和 USERSPACE2 重建一个可连接的数据库的话，那就很好了，这样用户很快就可以使用数据库。如果时间允许的话，可以再来恢复 USERSPACE3。下面的步骤展示了如何使用数据库重建实用程序来完成这一任务。

(1) 带 REBUILD 选项发出一个 RESTORE DATABASE 命令，指定只恢复部分表空间：

```
    db2 restore db test rebuild with tablespace
(SYSCATSPACE,USERSPACE1,USERSPACE2) taken at 20080516135136
```

虽然我们只想恢复 USERSPACE1 和 USERSPACE2，但是还必须恢复 SYSCATSPACE，因为这个表空间中存放着所有系统信息。没有它，DB2 就不知道关于这个数据库的结构的任何信息。

上面指定的目标镜像是包含 USERSPACE2 和 USERSPACE3 的镜像。这是包含我们要恢复的表空间的最近的备份。虽然 20080517135208 是三个备份当中最近的备份，但是我们不能使用它，因为它不包含 USERSPACE1、USERSPACE2 或 SYSCATSPACE。

下面的命令效果是一样的：

```
    db2 restore db test rebuild with all tablespaces in database except tablespace
       (USERSPACE3) taken at 20080516135136
```

(2) 带 TO END OF LOGS 选项发出一个 ROLLFORWARD DATABASE 命令：

```
    db2 rollforward db test to end of logs
```

(3) 带 STOP 选项发出一个 ROLLFORWARD DATABASE 命令：

```
    db2 rollforward db test stop
```

您可以选择前滚到另外一个时间点，而不是前滚到日志的最后。您所选择的时间点必须大于恢复中使用的备份镜像的时间戳。

至此，TEST 数据库已经可以连接，并且除了 USERSPACE3 之外的所有表空间都处于 NORMAL 状态。USERSPACE3 处于 RESTORE PENDING 状态。

您可以在以后通过一个常规的表空间恢复(不带 REBUILD 选项)来恢复 USERSPACE3。

(4) 发出一个 RESTORE DATABASE 命令，并指定要恢复的表空间：

```
    DB2 restore db test tablespace (USERSPACE3) taken at 20080517135208
```

(5) 带 TO END OF LOGS 选项发出一个 ROLLFORWARD DATABASE 命令，并指定

要前滚的表空间：

```
db2 rollforward db test to end of logs tablespace (USERSPACE3)
```

(6) 带 STOP 选项发出一个 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test stop
```

现在，TEST 数据库的所有 4 个表空间都处于 NORMAL 状态。

在生产环境中，或者在像前面那样的恢复场景中，只让一部分表空间快速恢复以对外提供可用性是很有用的。在测试环境中这样做也有用处，您可以只恢复感兴趣的那部分表空间。

7.8.4　使用包含日志文件的在线备份重建数据库

当重建一个可恢复数据库时，可以使用数据库备份，也可以使用表空间备份。备份可以是在线的，也可以是离线的。

如果您有一个包含日志文件的在线备份镜像，并且想使用这些日志来前滚数据库，那么可以使用 RESTORE DATABASE 命令的 LOGTARGET 选项从镜像中获取日志。

再次使用 TEST 数据库作为例子，假设备份镜像 TEST.3.DB2.NODE0000.CATN0000.20080517135208.001 是一个包含日志的在线备份镜像。要使用表空间备份和存储在备份镜像中的日志恢复整个数据库：

(1) 带 LOGTARGET 选项发出一个 RESTORE DATABASE 命令。在恢复期间，这些日志被提取到 LOGTARGET 指定的位置。

```
db2 restore db test rebuild with all tablespaces in database taken at
20080517135208   logtarget /logs
```

(2) 带 TO END OF LOGS 选项发出一个 ROLLFORWARD DATABASE 命令，并指定日志的位置：

```
db2 rollforward db test to end of logs overflow log path (/logs)
```

注意：

OVERFLOW LOG PATH 选项用于指定日志位置。

(3) 带 STOP 选项发出一个 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test stop
```

7.8.5 使用增量备份镜像重建可恢复数据库

增量备份镜像也可以用于重建数据库。当重建过程中涉及增量镜像时，默认情况下恢复实用程序会尝试为所有增量镜像使用自动增量恢复。如果没有使用 RESTORE DATABASE 命令的 INCREMENTAL AUTOMATIC 选项，但是目标镜像是一个增量备份镜像，那么恢复实用程序将使用自动增量恢复发出重建操作。

如果目标镜像不是一个增量镜像，但是另一个需要的镜像是增量镜像，那么恢复实用程序也将确保使用自动增量恢复来恢复那些增量镜像。不管是否指定了 INCREMENTAL AUTOMATIC 选项，恢复实用程序的行为都是一样的。

如果指定 INCREMENTAL 选项而没有指定 AUTOMATIC 选项，那么需要手动执行整个重建过程。恢复实用程序只是从目标镜像中恢复初始的元数据，就像在常规的手动增量恢复中一样。然后还需要使用所需的增量恢复链(见 7.4.3 节内容)来完成目标镜像的恢复。然后，为了重建数据库，还需要恢复剩下的镜像。这个过程可能比较冗长。

建议使用自动增量恢复来重建数据库。只有在恢复失败的情况下，才应该尝试通过手动方式重建数据库。

7.8.6 使用重定向选项重建可恢复数据库

由于重建功能是恢复实用程序的一部分，因此可以使用重定向方法重建数据库，就像在重定向恢复中那样。下面使用 REDIRECT 选项将整个 TEST 数据库重建到最近的时间点：

(1) 带 REBUILD 和 REDIRECT 选项发出一个 RESTORE DATABASE 命令：

```
    db2 restore db test rebuild with all tablespaces in database taken at
20080517135208    redirect
```

(2) 对要为之重新定义容器的每个表空间发出一个 SET TABLESPACE CONTAINERS 命令。例如：

```
    db2 set tablespace containers for 3 using (file '/newuserspace2' 10000)
    db2 set tablespace containers for 4 using (file '/newuserspace3' 15000)
```

(3) 带 CONTINUE 选项发出一个 RESTORE DATABASE 命令：

```
    db2 restore db test continue
```

(4) 带 TO END OF LOGS 选项发出一个 ROLLFORWARD DATABASE 命令(假设日志路径目录中所有日志都是可以访问的；否则，需要使用 OVERFLOW LOG PATH 选项来指定备选日志路径)：

```
db2 rollforward db test to end of logs
```

(5) 带 STOP 选项发出一个 ROLLFORWARD DATABASE 命令：

```
db2 rollforward db test stop
```

至此，这个数据库已经可以连接，并且所有表空间都处于 NORMAL 状态。

7.8.7　重建不可恢复数据库

到目前为止，我们讨论的所有重建方法都同样适用于不可恢复数据库。唯一的区别是：

- 如果一个数据库是不可恢复的，那么在重建操作中只能使用一个数据库备份作为目标镜像，因为不可恢复数据库没有可用的表空间备份。
- 当恢复完成时，马上就可以连接到数据库——不需要前滚操作。但是，任何尚未恢复的表空间都被置于 drop pending 状态，并且它们再也不能被恢复。

让我们看一个例子。假设有一个不可恢复的数据库 MYDB。MYDB 有 3 个表空间：SYSCATSPACE、USERSP1 和 USERSP2。在 20080521130000 做了一个完整数据库备份。

为了只使用 SYSCATSPACE 和 USERSP1 重建数据库：

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP1) taken at
20080521130000
```

在恢复之后，马上就可以连接数据库了。如果发出 LIST TABLESPACES 命令，您将看到 SYSCATSPACE 和 USERSP1 处于 NORMAL 状态，而 USERSP2 则处于 DROP PENDING 状态。现在，可以使用处于 NORMAL 状态的那两个表空间。

如果要做一个数据库备份，那么首先必须使用 DROP TABLESPACE 命令删除 USERSP2，否则备份会遭到失败。

7.8.8　数据库重建的限制

重建数据库的能力使恢复实用程序更加强大。但是，这方面也有一些限制：

- 重建的表空间中必须包括 SYSCATSPACE。
- 不能使用控制中心图形化工具执行重建操作。您只能使用命令行处理器(CLP)发出命令。
- 对于 9.1 版本之前的目标镜像，除非这个镜像是离线数据库备份，否则不能使用 REBUILD 选项。如果目标镜像是一个离线数据库备份，那么只有这个镜像中的表空间可以用于重建。在重建操作成功完成之后，需要迁移数据库。如果使用 9.1 版本之前的其他类型的目标镜像进行重建，那么将导致错误。

- 除非目标镜像是一个完整数据库备份，否则，如果目标镜像与被恢复的数据库在不同的操作系统中，那么不能对目标镜像发出 REBUILD 选项。如果目标镜像是一个完整数据库备份，那么只有这个镜像中的表空间可用于重建。

7.9 监控备份、复原和恢复进度

可以使用 LIST UTILITIES 命令来监控数据库上的备份、复原和前滚操作。

要对备份、复原和恢复操作使用进度监控，发出 LIST UTILITIES 命令并指定 SHOW DETAIL 选项：

```
list utilities show detail
```

以下是用于监控脱机数据库备份操作性能的输出的示例：

```
LIST UTILITIES SHOW DETAIL
    标识                          = 2
    类型                          =备份
   数据库名称                    = 样本
    描述                          = 脱机数据库
    开始时间                     = 10/30/2008 12:55:31.786115
    正在调速：
    优先级                        = 最低
    进度监控：
    估计完成百分比                = 41
            全部工作单元             = 20232453 个字节
            已完成的工作单元         = 230637 个字节
            开始时间               = 10/30/2008 12:55:31.786115
```

对于备份操作，将指定要处理的最初估计字节数。随备份操作进度更新要处理的字节数。显示的字节与镜像的大小不相等，不应该用作备份镜像大小的估计值。视镜像是增量备份还是压缩备份而定，实际镜像可能小很多。

对于复原操作，将不给定任何最初的估计值。而是指定 UNKNOWN。因为从镜像中读取每个缓冲区，所以将会更新实际读取的字节量。对于其中可能复原多个镜像的自动增量复原操作，将使用阶段来跟踪进度。每个阶段提供一个从增量链中复原的镜像。最初，只显示一个阶段。复原第一个镜像之后，将显示阶段的总数。在复原每个镜像时，将根据已处理的字节数来更新完成的阶段数。

对于崩溃恢复和前滚恢复，将有两个进度监控阶段：FORWARD 和 BACKWARD。FORWARD 阶段，读取日志文件并将日志记录应用于数据库。对于崩溃恢复，通过使用起

始日志序号至最后一个日志文件的结尾来估计工作总量。对于前滚恢复，当此阶段开始时，将工作总量估计值指定为 UNKNOWN。按字节计的已处理工作量将随进程的继续而更新。

在 BACKWARD 阶段，回滚 FORWARD 阶段任何未落实的更改。将提供按字节计的需处理日志数据量的估计值。按字节计的已处理工作量将随进程的继续而更新。

7.10　备份、恢复和复原期间表空间状态

表空间的当前状态由它的状态反映。与备份恢复相关的最常见表空间状态是：

- 备份暂挂。在前滚操作的某个时间点后，或不带有复制选项的装入操作后，表空间将置于此状态。在可使用该表空间之前必须对其备份(如果未进行备份就不能更新表空间，但允许只读操作)。
- 复原暂挂。如果取消了对表空间的前滚操作，或对表空间的前滚操作遇到了不可恢复错误(此时必须再次复原并前滚表空间)，会将表空间置于此状态。在复原操作期间，如果无法复原表空间，该表空间也会处于此状态。
- 正在前滚。表空间在对它进行的前滚操作正在进行中时，被置于此状态。一旦前滚操作成功完成，表空间就不再处于“正在前滚”状态。如果取消了对表空间的前滚操作，表空间也会结束此状态。
- 前滚暂挂。表空间在复原后发生了输入/输出(I/O)错误后被置于此状态。复原后，表空间可前滚到日志的末尾的某时间点。发生了 I/O 错误后，表空间必须前滚到日志的末尾。

7.11　优化备份、复原和恢复性能

在 DB2 V8 以后，在执行备份、恢复和复原操作时，DB2 将自动为缓冲区个数、缓冲区大小和并行性设置选择最佳值。这些值根据可用实用程序堆内存的数量、可用处理器数和数据库配置而定。因此，应根据系统上可用的内存大小，考虑通过增大 UTIL_HEAP_SZ 配置参数来分配更多内存。目的是在最大程度上减少完成备份、恢复和复原操作所需的时间。除非显式地输入以下命令参数的值，否则 DB2 将为它们选择一个值：

- WITH num-buffers BUFFERS
- PARALLELISM n
- BUFFER buffer-size

如果未指定缓冲区数和缓冲区大小而导致 DB2 自动设置这些值，那么对大型数据库的

影响应该最低。但是，对于小型数据库来说，会导致备份镜像大幅增大。即使写入磁盘的最后一个数据缓冲区只包含很少数据，也会将整个缓冲区写入镜像。在小型数据库中，这表示相当一部分的镜像可能为空。

还可以选择执行以下任何操作来缩短完成一次备份、恢复和复原操作所需的时间：

- 增大 PARALLELISM 参数的值，以使它反映正在操作的表空间数。

PARALLELISM 参数定义在压缩备份操作期间从数据库读取数据和压缩数据时，已启动的进程或线程数。将每个进程或线程分配给特定表空间，因此，为 PARALLELISM 参数指定的值大于要备份的表空间数并无益处。备份完此表空间后，它会请求另一个表空间。但是应注意：每个进程或线程都需要内存和 CPU 开销。

- 增加备份、恢复缓冲区大小。

默认的备份、恢复缓冲区大小在 DBM 配置参数中设置，如下所示：

```
db2 get dbm cfg | find /i “BUFSZ”
备份缓冲区默认大小(4KB)                    (BACKBUFSZ) = 1024
复原缓冲区默认大小(4KB)                    (RESTBUFSZ) = 1024
```

如果我们在备份、恢复和复原命令中没有显式地指明缓冲区大小，它们默认将使用上述 DBM 配置。

理想的备份、恢复缓冲区大小是表空间扩展数据块大小的倍数加一页。如果有多个扩展数据块大小不同的表空间，那么将值指定为扩展数据块大小的公倍数加一页。例如：

```
SYSCATSPACE         扩展数据块大小(页)              = 4
TEMPSPACE1          扩展数据块大小(页)              = 32
USERSPACE1          扩展数据块大小(页)              = 32
IBMDB2SAMPLEREL     扩展数据块大小(页)              = 32
DATA_SPACE          扩展数据块大小(页)              = 8
INDEX_SPACE         扩展数据块大小(页)              = 16
TS2                 扩展数据块大小(页)              = 32
```

在上面的例子中，我们有 7 个表空间，它们的扩展数据块大小不一样。这种情况下，如果我们要在备份、恢复和复原时设置备份、恢复缓冲区大小，应该设置成它们扩展数据块的公倍数。

在上面的例子中，32 是个公倍数。那么我们考虑设置备份、恢复缓冲区大小为 32+1=33 页。

- 增加缓冲区的数量。

使用的缓冲区至少是备份目标(或会话)的两倍，以确保备份目标设备无需等待数据。

- 使用多个目标设备备份恢复。

虽然在我们设置完上述参数后可以提高备份、恢复和复原性能，但是上述所有的设置都是基于 UTIL_HEAP_SZ 配置参数所配置的内存。所以在备份、恢复和复原期间应监控 UTIL_HEAP_SZ 内存的使用情况。请看下面的例子：

```
db2 get snapshot for database on sample
..............................略..............................................
数据库的内存使用情况:
  节点号                                    = 0
        内存池类型                          =备份/复原/实用程序堆
        当前大小(以字节计)                  = 17563648
        高水位标记(以字节计)                = 17563648
        已配置的大小(以字节计)              = 20512768
..............................略..............................................
```

在备份、恢复和复原期间监控数据库备份/复原/实用程序堆的当前大小和高水位标记。如果当前大小和高水位标记接近“已配置的大小”，并且我们内存资源充足，可以考虑增大 UITL_HEAP_SZ 参数。

7.12　备份恢复最佳实践

备份恢复策略其实就是在备份时间和恢复时间之间做一个选择折中和平衡。要保证数据的安全，可以使用以下最佳实践：

- 保证数据库处于归档日志模式，这样当数据库发生故障时便可以将它恢复到特定的时间点。
- 定期执行完整和递增的数据库备份。业务需求将最终决定时间表和频率。
- 如果数据库特别重要，考虑使用镜像日志。但是镜像日志会带来部分的性能开销。
- 根据业务需求和数据库大小指定合适的备份恢复策略。

在用户的备份恢复策略中要考虑以下事项：

- 考虑要使用的日志类型，即循环日志和归档日志。
- 决定对备份恢复操作采用的访问(读取)类型。
- 要意识到恢复动作可能包括前滚操作。
- 要清楚必须进行前滚操作的时间点。

表 7-5 总结归纳了在表空间的整个数据库级别上进行备份与恢复操作的各种考虑。

表 7-5　备份操作的限制与约束条件

	数据库离线备份		数据库在线备份	增量备份	表空间离线备份	表空间在线备份
日志类型	归档	循环	归档	归档	归档	归档
备份期间读取	不适用	不适用	可以存取	可以读取	不允许访问数据库	对数据库的充分存取
恢复之后的数据库状态	数据库处于前滚挂起状态	数据库处于一致性状态	数据库处于前滚挂起状态	数据库处于前滚挂起状态		数据库处于前滚挂起状态
前滚操作	在任何时刻	不适用	在备份结束后的任何时间点	备份结束后任何时间点	备份结束后任何时间点	在备份结束后的任何时间点

- 对于任务关键型数据库，在本地准备一个副本或备用数据库以供随时使用。
- 对于增量备份，确保各个备份文件的完备性。
- 尽量保留多份备份文件。一份放在本地以备数据库随时使用，一份放到磁带或带库上，另一份远程异地存放。
- 将活动日志文件和归档日志文件保存在不同的位置，并且各位置拥有充足的磁盘空间。
- 确保使用以下命令将数据库参数 **BLK_LOG_DSK_FUL** 设置为值 **YES**：

```
UPDATE DB CFG FOR db_name USING BLK_LOG_DSK_FUL YES
```

将 BLK_LOG_DSK_FUL 设置为 YES 后，当 DB2 遇到错误(如因为保存日志文件的文件系统变满)而导致应用程序挂起时，这样将允许我们解决错误，并允许完成运行中的事务。要解决日志磁盘空间不足错误，可以将原来的日志文件移动到另一个文件系统，或者扩大文件系统的空间。



DB2 故障诊断

首先要把“诊断”和“性能”这两个术语区分开(当然有的时候它们往往是有关联的)。问题诊断是数据库中最艰难的工作之一，例如，你的应用程序异常挂起了，你的应用程序时断时续等。如何定位问题所在呢？在一个复杂的系统中，应用程序、中间件、DB2 和操作系统每个环节都可能有问题，诊断会有很大的难度。本章主要讲解在 DB2 层面如何来进行数据库诊断。

本章主要讲解如下内容：

- DB2 故障诊断机制
- 故障诊断文件
- 故障诊断工具
- 故障诊断流程

8.1 DB2 故障诊断机制

8.1.1 故障诊断相关文件

FODC 表示首次故障数据捕获，它是 DB2 内部的故障处理机制(注：在 DB2 V9 之前，用到的是 FFDC——首次失败数据捕获，V9 后更名为 FODC)。它保留处理故障生成的信息，并将控制返回到受影响的引擎。捕获的数据保存在日志文件中，供问题分析之用。FODC 主要供 IBM Service 和 DBA 查看。它不仅仅应用在 DB2 产品中，也同样应用于 WebSphere Application Server、MQ 和 AIX 等 IBM 相关的产品中。运行时出现的事件和错误，信息一出现即加以捕获，并将其写入到日志文件中，供 DBA 进行分析。FODC 架构如图 8-1 所示。

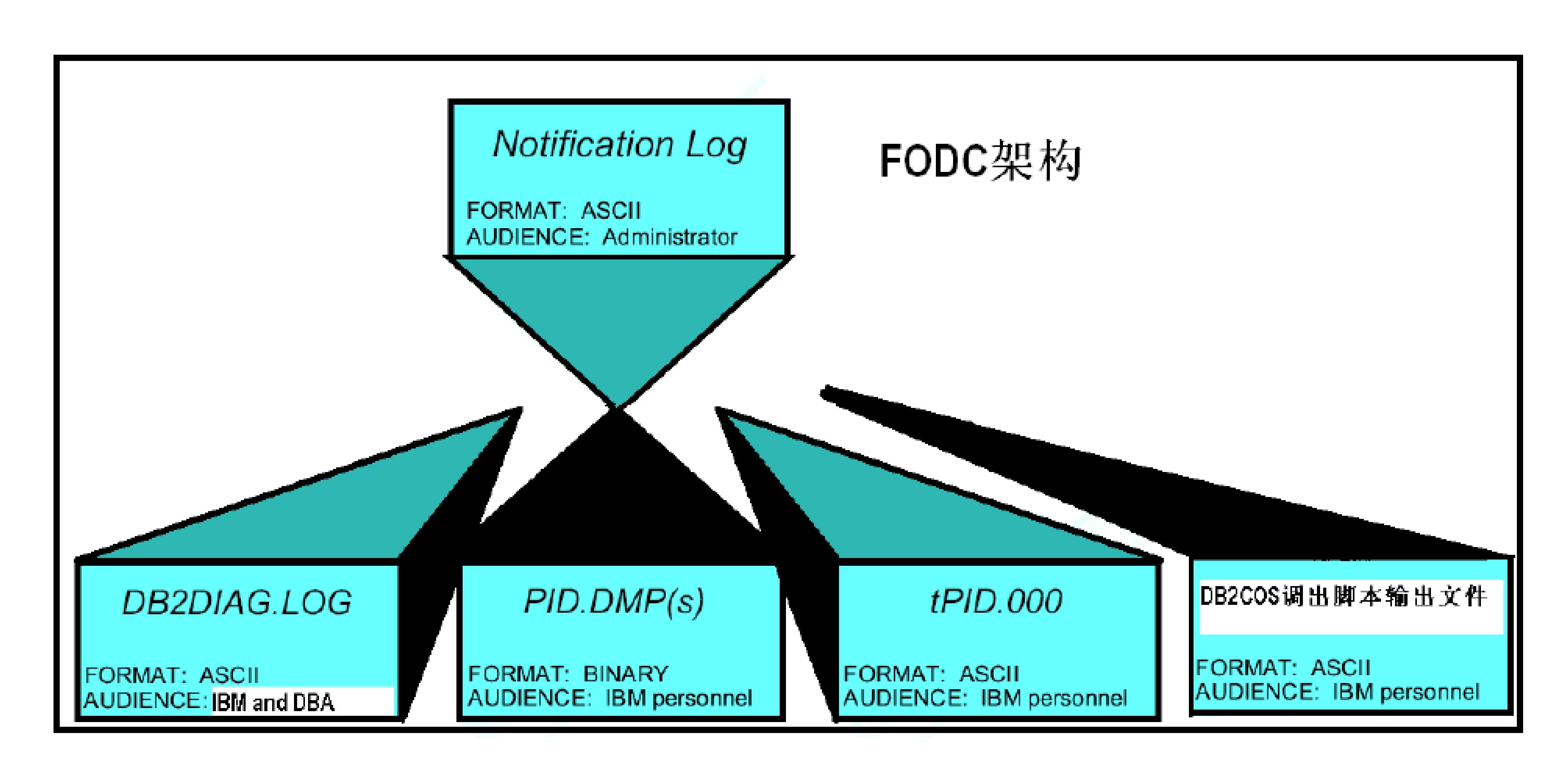


图 8-1 FODC 的架构

首次故障数据捕获(FODC)是用来捕获有关 DB2 数据库的基于场景的数据的过程。DB2 用户可根据特定症状手动调用 FODC，也可在检测到预定义场景或症状(例如，数据库挂起 hang)时自动调用 FODC。此信息减少了再现错误以获取诊断信息的需要。

在诊断路径 diagpath 下找到有关问题诊断的文件，这些文件中存放有关 FODC 信息。诊断路径可以通过下面的命令获取。也可更改 DBM 配置修改默认路径。

```
C:\>db2 get dbm cfg |  find /i "DIAG"
 诊断错误捕获级别                         (DIAGLEVEL) = 3
    诊断数据目录路径                          (DIAGPATH) = C:\
```

和诊断有关的 FODC 日志文件主要有以下几个：

管理通知日志(Notification Log)

发生重大事件时，DB2 将信息写入管理通知日志(“instance_name.nfy”)。该信息供数据库和系统管理员使用。通知消息提供了其他信息以补充提供的 SQLCODE。事件的类型和收集的信息的详细级别是由 NOTIFYLEVEL 配置参数确定的。

DB2 数据库管理器将下列类型的信息写入管理通知日志：DB2 实用程序(如 REORG 和 BACKUP)的状态、客户机应用程序错误、服务类更改、许可证发放活动、日志文件路径和存储问题、监视活动并为活动建立索引，以及表空间问题。数据库管理员可以使用这些信息来诊断问题、调整数据库或仅监视数据库。

管理通知日志消息还以标准化的消息格式记录到 db2diag.log。此日志只在 Linux/UNIX 上存在，在 Windows 平台此文件不存在。

DB2 诊断日志(“db2diag.log”)

此文本文件包含关于实例遇到的错误和警告的诊断信息。此信息用于故障诊断，且旨在供 DBA 使用。记录在此文件中的消息类型由 *diaglevel* 数据库管理器配置参数确定。

转储文件

对于某些错误情况，会将附加信息记录在以失败进程标识命名的二进制文件中。这些文件主要供 IBM 软件支持机构使用。原始堆栈转储可能包括在 ASCII 陷阱文件中。特定于数据库管理器内的组件的转储文件存储在相应 FODC 包目录中。

转储文件是在发生错误时创建的，它包含将有助于诊断问题的其他信息(例如内部控制块)。写至转储文件的每个数据项都具有与其相关联的时间戳记，以帮助进行问题确定。转储文件使用二进制格式，目的是供 IBM 内部人员来进一步定位是 bug 还是程序代码错误等等。这个文件需要用到 IBM 内部工具，所以一般我们是不能读的。

创建或追加转储文件时，会在 db2diag.log 中建立一个条目，指示所写数据的时间和类型。这些 db2diag.log 条目类似于以下所示：

```
2007-05-18-12.28.11.277956-240 I24861950A192 LEVEL: Severe
PID:1056930 TID:225448 NODE:000 Title: dynamic memory buffer
Dump File:/home/svtdbm5/sqllib/db2dump/1056930.225448.000.dump.bin
```

其实，这个文件对 DBA 而言基本上是不可读的，这个文件主要是在数据库出现重大问题时，我们把这个文件收集发送给 IBM 技术支持机构以判断是应用程序问题还是数据库 bug。这个问题必须使用 IBM 内部工具才能读，所以说虽然这个文件比较重要，不过通常我们很少用到。

陷阱(trap)文件

如果 DB2 由于陷阱、分段违例或异常而不能继续处理的话，它就会生成陷阱文件。DB2 接收到的所有信号或异常都会记录在陷阱文件中。陷阱文件还包含发生错误时正在运行的函数序列。此序列有时又称“函数调用堆栈”或“堆栈跟踪”。陷阱文件还包含有关捕获到信号或异常时进程的状态的其他信息。文件位于由 DIAGPATH 数据库管理器配置参数指定的目录中。陷阱文件包含如下信息：

- 可用虚拟内存量
- 发生陷阱时与产品的配置参数及注册表变量相关的值
- 发生陷阱时 DB2 产品使用的估计内存量
- 提供中断上下文的信息

在所有平台上，陷阱文件名以进程标识(PID)开头，后跟线程标识(TID)，再跟分区号(在单分区数据库上为 000)，并以“.trap.txt”结尾。

还有一些诊断陷阱，它们是在发生某些特定条件(这些条件不一定会使实例崩溃)时由代码生成的，在查看堆栈时非常有用。这些陷阱是使用 PID 以十进制格式命名的，后跟分区号(在单分区数据库中为 0)。

例如：

- 6881492.2.000.trap.txt 是进程标识(PID)为 6881492，线程标识(TID)为 2 的陷阱文件。
- 6881492.2.010.trap.txt 是进程和线程在分区 10 上运行的陷阱文件。

可将 db2pd(这个命令我们后面会讲解)命令与-stack all 或-dump 选项配合使用，以根据需要生成陷阱文件。目的是供 IBM 内部人员来进一步定位是 bug 还是程序代码错误等等。其实，这个文件和转储文件一样，对 DBA 而言基本上是不可读的，这个文件主要是在数据库出现重大问题时，我们把这个文件收集发送给 IBM 技术支持机构以判断是应用程序问题还是数据库 bug。这个问题必须使用 IBM 内部工具才能读，所以说虽然这个文件比较重要，不过通常我们很少用到。

DB2 调出脚本(db2cos)输出文件

db2cos 脚本(db2 callout script)是 DB2 V8.2 以后提供给我们用于高级复杂诊断的一个脚本，DB2 V8.2 版本之前，DB2 在故障诊断方面没有什么好的工具，只能查看 db2diag.log。将 DB2 V8.2 之后出现的 db2cos 脚本和 db2pd 结合起来可以为 DBA 提供一个强大的故障诊断工具。在默认情况下，当数据库管理器因为应急启动、陷阱、分段违例或异常而不能继续处理时，将会自动调用 db2cos 脚本。每个默认 db2cos 脚本将调用 db2pd(关于这个强大的工具，我们本章后面会讲解)命令以打开方式收集信息。db2cos 脚本的名称的格式为 db2cos_hang、db2cos_trap 等等。每个脚本的行为方式类似，但 db2cos_hang 有所不同，它是通过 db2fodc 工具调用的。默认 db2cos 脚本将调用 db2pd 命令以打开方式收集信息。根据 db2cos 脚本中包含的命令，db2cos 输出文件的内容会有所不同。

默认 db2cos 脚本在 bin 目录中。在 UNIX 操作系统上，此目录是只读的。可将 db2cos 脚本复制至 adm 目录，必要时可在该位置修改该文件。如果 db2cos 脚本在 adm 目录中，那么会运行该脚本，否则会运行 bin 目录中的脚本。

Windows 上默认 db2cos.bat 的内容：

```
setlocal
:iterargs
if %0. == . goto iterdone
   if /i %0. == INSTANCE. set INSTANCE=%1
   if /i %0. == DATABASE. set DATABASE=%1
```

```
    if /i %0. == TIMESTAMP. set TIMESTAMP=%1
    if /i %0. == APPID. set APPID=%1
    if /i %0. == PID. set PID=%1
    if /i %0. == TID. set TID=%1
    if /i %0. == DBPART. set DBPART=%1
    if /i %0. == PROBE. set PROBE=%1
    if /i %0. == FUNCTION. set FUNCTION=%1
    if /i %0. == REASON. set REASON=%1
    if /i %0. == DESCRIPTION. set DESCRIPTION=%1
    if /i %0. == DiAGPATH. set DIAGPATH=%1
    shift
goto iterargs
:iterdone
if %DATABASE%. == . goto no database
    db2pd -db %DATABASE% -inst >> %DIAGPATH%\db2cos%PID%%TID%.%DBPART%
    goto exit
:no_database
    db2pd -inst >> %DIAGPATH%\db2cos%PID%%TID%.%DBPART%
:exit
```

通过上面的脚本可以看到，对于数据库级的事件或故障，默认的 db2cos 脚本用-db 和 -inst 选项调用 db2pd。

还可通过 db2pdcfg -cos 命令来配置触发 db2cos 调用的信号类型。默认配置用于要在发生应急启动或陷阱时运行的 db2cos 脚本。但是，在默认情况下，生成的信号不会启动 db2cos 脚本。例如下面的使用 db2pdcfg 收集锁超时的配置。

为了每当出现锁超时时启动 db2cos 脚本，DBA 调用 db2pdcfg 实用程序，如下所示：

```
db2pdcfg -catch locktimeout count=1 --使用 db2pdcfg 配置 db2cos 脚本的调用
```

-catch 选项指定应该自动导致调用 db2cos 脚本的故障或事件。对于锁超时事件，可以指定字符串 locktimeout。或者，可以指定与锁超时相应的 SQL 错误码和原因码：

```
db2pdcfg -catch 911,68 count=1  --用于捕捉锁超时的另一种 db2pdcfg 调用
```

发生应急启动、陷阱、分段违例或异常时，事件顺序如下所示：

(1) 创建陷阱文件

(2) 调用信号处理程序

(3) 调用 db2cos 脚本(取决于启用的 db2cos 设置)

(4) 在管理通知日志中记录相应条目

(5) 在 db2diag.log 中记录相应条目

db2cos 脚本中的 db2pd 命令收集到的默认信息包括有关操作系统、已安装 DB2 产品的

版本和服务级别、数据库管理器和数据库配置的详细信息，以及有关以下各项的状态的信息：代理程序、内存池、内存块、应用程序、实用程序、事务、缓冲池、锁定、事务日志、表空间和容器。此外，它还会提供有关下列各项的信息：动态、静态和目录高速缓存的状态、表和索引统计信息、恢复状态以及重新优化的 SQL 语句及活动语句列表。如果需要收集进一步的信息，那么只需使用附加命令更新 db2cos 脚本。

调用默认 db2cos 脚本时，它将在 DIAGPATH 数据库管理器配置参数指定的目录中生成输出文件。这些文件名为 XXX.YYY.ZZZ.cos.txt，其中 XXX 是进程标识(PID)，YYY 是线程标识(TID)，而 ZZZ 是数据库分区号(对于单分区数据库则为 000)。如果存在多线程陷阱，那么会对每个线程单独调用 db2cos 脚本。如果 PID 和 TID 组合多次出现，那么该数据将追加至文件。还会显示时间戳记，所以您可以区分输出的迭代。

8.1.2 收集故障诊断信息

发生影响实例的中断时，可自动将诊断信息收集到包中。包中的信息也可手动创建。

自动收集诊断信息

在出现故障时，数据库管理器调用 db2fodc 命令以自动执行首次故障数据捕获(FODC)。

为了使中断与 DB2 诊断日志和其他故障诊断文件相关，会同时将诊断消息写至管理通知日志和 db2diag.log。诊断消息包括 FODC 目录名和创建 FODC 目录时的时间戳记。FODC 包描述文件在新的 FODC 目录中。

- 配置诊断信息的自动收集

必须先指示数据库管理器要采取的操作，数据库管理器才能自动执行操作。

系统会设置一些标志，来指示数据库操作期间遇到错误或警告时数据库管理器应采取的操作。要采取的操作包括：

  - ◊ 在 db2diag.log 中生成堆栈跟踪(默认)
  - ◊ 运行标注脚本 db2cos(默认)
  - ◊ 停止跟踪命令(db2trc)

- 更改首次故障数据捕获(FODC)选项

使用“配置 DB2 数据库的问题确定行为”(db2pdcfg)命令来更改首次故障数据捕获(FODC)选项。FODC 选项是使用 db2pdcfg 工具在 DB2FODC 注册表变量中设置的。这些选项会影响 FODC 情况中有关数据捕获的数据库系统行为。

诊断信息的手动收集

首次故障数据收集命令(db2fodc)用于收集有关可能中止的信息或出现严重性能问题时的

信息。运行 db2fodc 命令后，会在当前诊断路径下自动创建新目录 FODC_hang_<timestamp>。会运行 db2cos_hang 脚本。此脚本控制收集并放在 FODC 子目录中的数据集合。FODC 子目录是否存在取决于 db2fodc 命令的运行方式或 DB2 注册表变量的配置。

FODC 数据及其放置

diagpath 配置参数用来指定一个目录，它将包含根据 *diaglevel* 参数的值可能会生成的错误文件、事件日志文件、警报日志文件以及任何转储文件。

根据您的平台，此目录可能包含转储文件、陷阱文件、错误日志、通知文件、警报日志文件和"首次故障数据集合"程序包。

如果此参数为 Null，那么诊断信息将写入下列其中一个目录或文件夹中的文件：

- 在 Windows 环境中：用户数据文件(例如实例目录下的文件)被写入不同于安装了代码的位置，例如 C:\Documents and Settings\All Users\Application Data\IBM\DB2*Copy Name*，其中 *Copy Name* 是 DB2 副本的名称。
- 在 Linux 和 UNIX 环境中：信息将写入 *INSTHOME*/sqllib/db2dump，其中 *INSTHOME* 是实例的主目录。

根据实例内的中断类型，首次故障数据捕获(FODC)会导致创建子目录及收集的特定内容。将创建一系列子目录及文件和日志的集合。

注意：
应定期清理转储目录以防止它变得太大。

8.1.3　设置故障诊断级别

设置管理通知日志文件的错误捕获级别

在 UNIX 平台上，管理通知日志是称为 *instance*.nfy 的文本文件。在 Windows 上，所有管理通知消息都写到"事件日志"中。错误可由 DB2、"运行状况监视器"、Capture 和 Apply 程序，以及用户应用程序写入。DB2 记录在管理通知日志中的信息由 NOTIFYLEVEL 设置确定。

- 要检查当前设置，发出 GET DBM CFG 命令。

查找以下变量：

```
通知级别(NOTIFYLEVEL)                        = 3
```

- 要改变该设置，使用 UPDATE DBM CFG 命令。例如：

```
DB2 UPDATE DBM CFG USING NOTIFYLEVEL X
```

其中 X 是期望的通知级别。

X 的有效值为：

◊ 0—— 未捕获任何管理通知消息(不建议使用此设置)

◊ 1—— 致命或不可恢复错误。仅记录致命和不可恢复错误。要从这些情况中的某些情况进行恢复，可能需要来自 DB2 服务机构的帮助

◊ 2—— 需要立即操作。记录了需要系统管理员或数据库管理员立即注意的情况。如果未解决该情况，那么它可能会导致致命错误。非常重要并且没有错误的活动(例如恢复)的通知可能也在此级别记录。此级别将捕获“运行状况监视器”警报

◊ 3—— 重要信息，不需要立即操作。记录没有威胁也不需要立即操作但是可能表示并非最佳系统的情况。此级别将捕获“运行状况监视器”警报、“运行状况监视器”警告和“运行状况监视器”引起注意信息

◊ 4—— 记录最全的信息

管理通知日志包括具有最多且包括 *notifylevel* 的值的消息。例如，将 *notifylevel* 设置为 3 将导致管理通知日志包括可应用于级别 1、2 和 3 的消息。

建议：

您可能希望增大此参数的值以收集更多问题确定数据来帮助解决问题。注意，您必须将 *notifylevel* 的值设置为 2 或更高，以便“运行状况监视器”向在其配置中定义的联系人发送所有通知。

设置诊断日志文件错误捕获级别

DB2 诊断日志是包含 DB2 记录的文本信息的文件。此信息可用于故障诊断，但主要用于 DB2 客户支持。DB2 在 db2diag.log 中记录的信息由 DIAGLEVEL 设置确定。

● 要检查当前设置，发出 GET DBM CFG 命令。

查找以下变量：

```
诊断错误捕获级别(DIAGLEVEL)                    = 3
```

要动态更改该值，请使用 UPDATE DBM CFG 命令。

● 要联机更改数据库管理器配置参数，请使用以下命令：

```
db2 update dbm cfg using <parameter-name> <value>
```

例如：

```
DB2 UPDATE DBM CFG USING DIAGLEVEL X
```

其中 X 是期望的通知级别。

如果要诊断可再现的问题，支持人员建议您在执行故障诊断时使用 DIAGLEVEL 4。

X 的有效值为：

◊ 0——没有捕获到诊断数据
◊ 1——仅严重错误
◊ 2——所有错误
◊ 3——所有错误和警告
◊ 4——所有错误、警告以及参考消息

建议：

在应用运行初期为了仔细定位问题，可以考虑把诊断级别设置为 4 以收集详细信息，等应用运行稳定后再把诊断级别调低。

8.2　深入讲解故障诊断文件

8.2.1　解释管理通知日志文件条目

使用文本编辑器来查看怀疑发生了问题的机器上的管理通知日志文件。记录的最新事件在文件的最后面。通常，每个条目包含下列部分：

- 时间戳记
- 报告错误的位置。应用程序标识允许您匹配在服务器和客户机的日志上与应用程序有关的条目
- 说明错误的诊断消息(通常以“DIA”或“ADM”开头)
- 任何可用的支持数据(例如 SQLCA 数据结构)和指向任何其他转储文件或陷阱文件的位置的指针。

管理日志与数据库中的所有日志一样会不断增长。根据每个文件中记录的内容不同，某些日志的增长速度会比其他日志的增长速度快。日志过大时，应对其进行备份并清除。下一次系统需要新日志时会自动生成。

以下示例显示样本日志条目的标题信息，且标识了日志的所有部分。

注意：

不是所有日志条目都包括所有这些部分。

2008-02-15-19.33.37.630000 1 实例：DB2 2 节点：000 3
PID：940(db2syscs.exe) TID：660 4 Appid：*LOCAL.DB2.020205091435 5
恢复管理器 6 sqlpresr 7 探测点：1 8 数据库：SAMPLE 9
ADM1530E 10 已启动崩溃恢复。11

1 消息的时间戳记。

2 生成该消息的实例的名称。

3 对于多分区系统，此项为生成该消息的数据库分区。在非分区数据库中，该值为“000”。

4 进程标识(PID)，后跟进程名称，再后跟负责生成消息的线程标识(TID)。

5 进程正在为其工作的应用程序的标识。在本示例中，生成消息的进程正在为标识为*LOCAL.DB2.020205091435 的应用程序工作。

要标识关于特定应用程序标识的信息，执行下列其中一项操作：

- 在 DB2 服务器上使用 LIST APPLICATIONS 命令来查看应用程序标识列表。可以根据此列表确定有关遇到错误的客户机的信息，例如其节点名以及 TCP/IP 地址。
- 使用 GET SNAPSHOT FOR APPLICATION 命令查看应用程序标识列表。

6 写入消息的 DB2 组件。对于由使用 db2AdminMsgWrite API 的用户应用程序编写的消息，该组件将读取“用户应用程序”。

7 提供消息的函数的名称。此函数在写入消息的 DB2 组件中运行。对于由使用 db2AdminMsgWrite API 的用户应用程序编写的消息，该函数将读取“用户函数”。

8 唯一内部标识。此数字允许 DB2 客户支持和开发在报告了消息的 DB2 源代码中找到相应位置。

9 发生错误的数据库。

10 以十六进制代码指示错误类型和编号的消息(如果存在)。

11 说明记录的事件的消息文本(如果存在)。

8.2.2 解释诊断日志文件条目

使用 db2diag.log 分析工具(db2diag)来过滤并格式化 db2diag.log 文件。虽然已使用标准化消息格式将管理通知日志消息记录至 db2diag.log，但还是建议先查看 db2diag.log 以了解数据库所发生的情况。

除了使用 db2diag 之外，还可使用文本编辑器来查看怀疑发生了问题的机器上的诊断日志文件。记录的最新事件在文件的最后面。

注意：

管理和诊断日志会不断增加。当它们变得太多时，对其进行备份然后清除。下一次系

统需要它们时，会自动生成一组新的文件。

以下示例显示样本日志条目的标题信息，且标识了日志的所有部分。

注意：

不是所有日志条目都包括所有这些部分。只有开头的一些字段(时间戳记至 TID)和函数才会显示在所有 db2diag.log 记录中。

```
2008-05-18-14.20.46.973000-240 1 I27204F655 2 LEVEL: Info 3
PID : 3228 4 TID : 8796 5 PROC : db2syscs.exe 6
INSTANCE: DB2MPP 7 NODE : 002 8 DB : WIN3DB1 9
APPHDL : 0-51 10 APPID: 9.26.54.62.45837.080518182042 11
AUTHID : UDBADM 12
EDUID : 8796 13 EDUNAME: db2agntp 14 (WIN3DB1) 2
FUNCTION: 15 DB2UDB, data management, sqldInitDBCB, 16 probe:4820
DATA #1 : 17 String, 26 bytes
Setting ADC Threshold to:
DATA #2 : unsigned integer, 8 bytes
1048576
```

1 消息的时间戳记和时区。

2 记录标识字段。对于创建 DB2 诊断日志的平台，db2diag.log 的记录标识指定要记录的当前消息的文件位移(如“27204”)和消息长度(如“655”)。

3 与错误消息相关联的诊断级别。例如参考、警告、错误、严重或事件。

4 进程标识。

5 线程标识。

6 进程名称。

7 生成该消息的实例的名称。

8 对于多分区系统，此项为生成该消息的数据库分区。在非分区数据库中，该值为“000”。

9 数据库名称。

10 应用程序句柄。此值与 db2pd 输出和锁定转储文件中使用的值相对应。它包括后跟协调程序索引编号并且用破折号分开的协调程序分区号。

11 进程正在为其工作的应用程序的标识。在本示例中，生成消息的进程正在为标识为 9.26.54.62.45837.070518182042 的应用程序工作。

TCP/IP 生成的应用程序标识由 3 个部分组成：

- IP 地址：它表示为 32-bit 位数字，最大显示为 8 位十六进制字符。
- 端口号：显示为 4 位十六进制字符。
- 此应用程序的实例的唯一标识。

注意：

如果 IP 地址或端口号的十六进制版本以 0 至 9 开头，那么它们将分别转换为 G 至 P。例如，“0”映射为“G”，“1”映射为“H”，依此类推。IP 地址 AC10150C.NA04.006D07064947 表示为如下所示：IP 地址仍为 AC10150C，它将转换为 172.16.21.12。端口号为 NA04。第一个字符为“N”，它将映射为“7”。因此，端口号的十六进制为 7A04，将它转换为十进制格式为 31236。

要标识关于特定应用程序标识的信息，执行下列其中一项操作：

- 在 DB2 服务器上使用 LIST APPLICATIONS 命令来查看应用程序标识列表。可以根据此列表确定有关遇到错误的客户机的信息，例如其数据库分区名以及 TCP/IP 地址。
- 使用 GET SNAPSHOT FOR APPLICATION 命令查看应用程序标识列表。
- 使用 db2pd -applications -db <sample>命令。

12 授权标识。

13 引擎可调度单元标识。

14 引擎可调度单元的名称。

15 产品名(“DB2”)、组件名(“数据管理”)、写入消息的函数名(“sqlInitDBCB”)以及函数内的探测点(“4820”)。

16 DB2 内部组件，主要看第 4 个字母，例如 sqldInitDBCB 的第 4 个字母为 d，表示是数据管理。其他的组件如图 8-2 所示。关于这些内部组件的详细讲解请参见《深入解析 DB2》一书。

Fourth Position Letters	Type of Activity indicated
b	Buffer pool management and manipulation
c	Communications between clients and servers
d	Data management
e	Database engine processes
o	Operatiing system calls(such as opening and closing files)
p	Data protection(such as locking and logging)
r	Relational database services
s	Soring operations
x	Indexing operations

图 8-2　DB2 的内部组件

17 被调用函数返回的信息。可能会返回多个数据字段。

8.3　故障诊断工具

8.3.1　使用 db2support 收集环境信息

对 DB2 问题收集信息时，您需要运行的最重要的 DB2 实用程序是 db2support。db2support 实用程序用于自动收集所有可用的 DB2 诊断信息和系统诊断信息，如图 8-3 所示。它还有一个可选的交互式“问与答”会话，该会话会提出有关问题的详情。

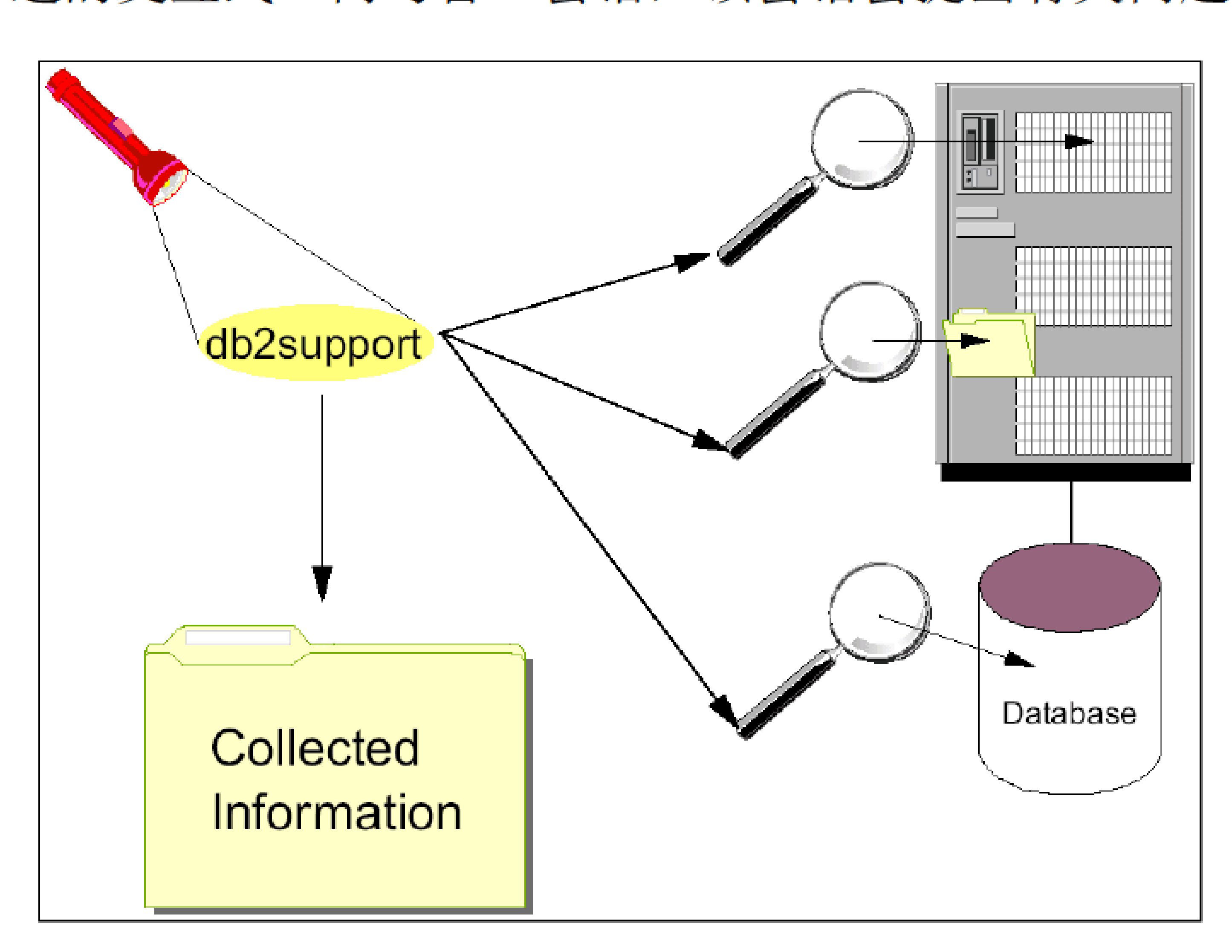


图 8-3　使用 db2support 收集环境信息

如果使用 db2support 实用程序来帮助将信息传送至 IBM 支持机构，那么在系统遇到问题时运行 db2support 命令，该工具就会及时地收集信息，例如操作系统性能详细信息等。即使在出现问题时无法运行实用程序，您仍可以在问题停止之后发出 db2support 命令，因为会自动生成某些首次故障数据捕获(FODC)诊断文件。

db2support 捕获的信息类型取决于调用命令的方式、是否启动了数据库管理器以及能否连接至数据库。对于调试问题所需的大多数信息的收集而言，以下基本调用通常已经足够(注意，如果使用-c 选项，那么该实用程序将建立与数据库的连接)：

```
db2support <output path> -d <database name> -c
db2support <directory> -d sample -s -m
```

输出的收集非常方便，并且会存储在压缩的 ZIP 归档 db2support.zip 中，以便可以很轻

松地在任何系统上对其进行传送和解压缩。

db2support 命令收集到的默认信息包括有关操作系统、已安装 DB2 产品的版本和服务级别、数据库管理器和数据库配置的详细信息，以及有关以下各项的状态的信息：db2diag.log 所有陷阱文件、锁定列表文件、转储文件、各种与系统有关的文件、各种系统命令的输出、db2cli.ini、活动日志文件、缓冲池和表空间(SQLSPCS.1 和 SQLSPCS.2)控制文件、db2dump 目录的内容、扩展系统信息、日志文件头文件、恢复历史记录文件等几乎所有用到的信息。

8.3.2 db2ls 和 db2level

db2ls

由于能够在系统上安装 DB2 产品的多个副本，并且能够灵活地在您选择的路径中安装 DB2 产品和功能部件，所以需要使用一个工具来帮助跟踪已经安装了哪些 DB2 产品及其安装位置。在受支持的 Linux 和 UNIX 操作系统上(注：在 Windows 操作系统上*不能*使用 db2ls 命令)，db2ls 命令将列示安装在系统上的 DB2 产品和功能部件。可以使用 db2ls 命令来列示 DB2 产品在系统上的安装路径、DB2 产品级别、级别、修订包、特殊安装编号、安装日期、安装程序用户标识等。

db2ls 命令是可用来查询 DB2 产品的唯一方法。您不能使用 Linux 或 UNIX 操作系统本机实用程序(例如 pkginfo、rpm、SMIT 或 swlist)来查询 DB2 产品。包含用于查询 DB2 安装并与其进行交互的本机安装实用程序的任何现有脚本将需要进行更改。

要列示 DB2 产品在系统上的安装路径和 DB2 产品级别，请输入以下命令：

```
# cd /usr/local/bin
# ./db2ls
Install Path Level Fix Pack Special Install Number Install Date
-----------------------------------------------------------------
/opt/IBM/db2/V9.1_01 9.1.0.0 0 Mon Sep 10 22:40:45 2007 CDT
```

该命令将列示安装在系统上的每个 DB2 产品的下列信息：

```
db2ls -q -p -b baseInstallDirectory  --要列示有关特定安装路径中的 DB2 产品或功能部件的信息，必须指定 q 参数
```

其中：

- *q* 指定您要查询的产品或功能部件。此参数是必需的。如果查询了 DB2 版本 8 产品，那么会返回空白值。
- *p* 指定列表显示产品而不是显示功能部件。

● *b* 参数指定产品或功能部件的安装目录。

db2level

db2level 命令帮助您确定 DB2 实例的版本和服务级别(构建级别和修订包级别)。

在 Windows 系统上运行 db2level 命令的典型结果为：

```
C:\>db2level
DB21085I 实例 "DB2" 使用"32"位和 DB2 代码发行版 "SQL09050"，级别标识为
"03010107"。
参考标记为"DB2 v9.5.0.808"、"s071001"和"NT3295"，修订包为"0"。
产品使用 DB2 副本名 "DB2COPY1" 安装在"C:\PROGRA~1\IBM\SQLLIB" 中。
```

这 4 个参考标记的组合唯一地标识 DB2 实例的精确服务级别。在定位问题以及联系 IBM 确认产品 bug 时，此信息非常重要。

对于 JDBC 或 SQLJ 应用程序，如果使用的是用于 SQLJ 和 JDBC 的 DB2 驱动程序，那么可通过运行 db2jcc 实用程序来确定驱动程序的级别：

```
C:\>db2jcc -version
IBMDB2JDBC Universal Driver Architecture 3.50.152
```

8.3.3　使用 db2diag 分析 db2diag.log 文件

db2diag 工具用于对 db2diag.log 中提供的大量信息进行过滤和格式化。过滤 db2diag.log 记录可缩短诊断问题时查找所需记录的时间。过去在没有这个命令之前，对于 DBA 来说读取 db2diag.log 文件通常是非常痛苦的，因为很多时候客户没有清理 db2diag.log 文件，所以我们不得不频繁地使用 grep 来过滤文件内容。DB2 V8 以后增加了 db2diag 命令，可以很方便地对诊断日志格式化、过滤和解释。下面我们举一些使用 db2diag 的示例。

例 8-1　按数据库名称过滤 db2diag.log。

如果实例中有若干数据库，并且您只希望显示与数据库“SAMPLE”有关的消息，那么可以按如下所示过滤 db2diag.log：

```
db2diag -g db=SAMPLE
```

因此，将仅显示包含“DB=SAMPLE”的 db2diag.log 记录：

```
2008-02-15-19.31.36.114000-300 E21432H406         级别：错误
PID：940                  TID：660         PROC：db2syscs.exe
实例：DB2                    节点：000         数据库：SAMPLE
APPHDL：0-1056                 APPID：*LOCAL.DB2.060216003103
函数：DB2 UDB，基本系统实用程序，sqleDatabaseQuiesce，探测点：2
```

```
消息：ADM7507W  数据库停顿请求成功完成。
```

例 8-2 按进程标识过滤 db2diag.log。

以下命令可用来显示进程标识(PID)为 2200，并且在分区 0、1、2 或 3 上运行的进程生成的所有严重错误消息：

```
db2diag -g level=Severe,pid=2200 -n 0,1,2,3
```

注意，此命令可能以不同方式写入，包括 db2diag -l severe -pid 2200 -n 0,1,2,3。还要注意的是，-g 选项指定区分大小写的搜索，所以此处"Severe"会起作用，但如果使用了"severe"则会失败。满足如下要求时，这些命令将成功检索 db2diag.log 记录：

```
2008-02-13-14.34.36.027000-300 I18366H421        级别：严重
PID     : 2200                  TID  : 660        PROC : db2syscs.exe
实例：DB2                        节点：000          数据库：SAMPLE
APPHDL：0-1433                  APPID：*LOCAL.DB2.060213193043
函数：DB2 UDB，数据管理，sqldPoolCreate，探测点：273
返回码：ZRC=0x8002003C=-2147352516=SQLB_BAD_CONTAINER_PATH “错误的容器路径”
```

例 8-3 格式化 db2diag 工具输出。

以下命令过滤 2008 年 5 月 1 日之后发生，并且包含分区 0、1 或 2 上记录的所有非严重错误和严重错误的所有记录。它会输出匹配的记录，因此时间戳记、分区号和级别将出现在第一行上，而 pid、tid 和实例名将出现在第二行上，之后是错误消息：

```
db2diag -time 2008-05-01 -node "0,1,2" -level "Severe, Error" |
db2diag -fmt "Time: %{ts}
分区：%node Message Level: %{level} \nPid: %{pid}  Tid: %{tid}
实例：%{instance}\nMessage: @{msg}\n"
```

生成的输出示例如下所示：

```
时间：2008-05-15-19.31.36.099000 分区：000 消息级别：错误
Pid：940 Tid：40 实例：DB2
消息：ADM7506W 已经请求了数据库停顿。
```

例 8-4 过滤来自不同工具的消息。

下列示例显示如何仅查看来自数据库管理器中的特定工具(或所有工具)的消息。受支持的工具包括：

- ALL，这会返回来自所有工具的记录 。
- MAIN，这会返回来自 DB2 常规诊断日志的记录，如 db2diag.log 和管理通知日志。
- OPSTATS，这会返回与优化器统计信息有关的记录。

要读取来自 MAIN 工具的消息，请使用以下命令：

```
db2diag -facility MAIN
```

要显示来自 OPSTATS 工具的消息并过滤出级别为 Severe 的记录，请使用以下命令：

```
db2diag -fac OPSTATS -level Severe
```

要显示来自所有可用工具的消息并过滤出实例为 harmistr，级别为 Error 的记录，请使用以下命令：

```
db2diag -fac all -g instance=harmistr,level=Error
```

要显示 OPSTATS 工具中级别为 Error，并且以特定格式输出时间戳记和 PID 字段的所有消息，请使用以下命令：

```
db2diag -fac opstats -level Error -fmt " Time :%{ts} Pid :%{ts}"
```

例 8-5　下面例子显示如何使用 db2diag 命令解释 DB2 跟踪或 db2diag.log 中的返回码。

```
E:\db2v9\BIN>db2diag -rc fffffb95
Input ECF string 'fffffb95' parsed as 0xFFFFFB95 (-1131)。
NOTE: ../sqz/sqlzwhatisrc.C:
V7 input ZRC 0xFFFFFB95 (-1131) may translate to V8 ZRC value of 0x83000
B95 (-2097149035)
ZRC value to map: 0x83000B95 (-2097149035)
V7 Equivalent ZRC value: 0xFFFFBB95 (-17515)
ZRC class :
Unexpected Operating System error (Class Index: 3)
Component:
Unknown component (Component Index: 0)
Undefined as ofDB2v8.1.14.292; s061108
Attempting to lookup value 0xFFFFFB95 (-1131) as a sqlcode Sqlcode -1131
SQL1131N  DARI (Stored Procedure) process has been terminated abnormally.
```

例 8-6　查找应用程序句柄 APPHDL 为 0-222 的所有诊断日志条目：

```
db2diag -g APPHDL="0-222"
```

例 8-7　查找应用程序句柄 APPHDL 为 0-222，在分区 0 上的所有诊断日志条目：

```
db2diag -g APPHDL="0-222",NODE=000
```

例 8-8　查找进程 1060946 的所有严重错误(Severe)：

```
db2diag -g PID=1060946,LEVEL=Severe
```

例 8-9 查找所有 FUNCTION 名称中包含 fetch 的诊断日志条目：

```
db2diag -g FUNCTION:=fetch
```

例 8-10 查找所有 component 名称以"base sys"开头的诊断日志条目：

```
db2diag -g "COMPONENT^=base sys"
```

例 8-11 查找所有返回码为"ZRC=0x80120086"的记录：

```
db2diag -g RETCODE:=0x80120086
```

除了过滤查找之外，db2diag 还可以格式化输出。您可以指定查找结果的输出格式。关于格式化输出的详细帮助，请使用"db2diag -h fmt"命令查看。下面简单介绍一个例子：

```
db2diag -time 2007-12-22 -node "0,1,2" -level "Severe, Error" |db2diag -fmt
"Time: %{ts} Partition: %node Message Level:%{level} \nPid: %{pid} Tid: %{tid}
Instance:%{instance}\nMessage: @{msg}\n"
```

该命令将查找 2007 年 12 月 22 日以来在分区 0、1、2 上，错误级别为 Severe 和 Error 的错误，并按照下面的格式输出：

```
Time : 2007-12-28-14.32.01.067843 Partition : 000 Message Level :Error
Pid : 1871948 Tid : 1 Instance :db2inst1
Message : ZRC=0x860F000A=-2045837302=SQLO FNEX "File not found."
DIA8411C A file "" could not be found.
```

db2diag 工具非常强大，有关更多信息，请发出下列命令查看：

- db2diag -help 提供所有可用选项的简短描述
- db2diag -h brief 提供对所有不带示例的选项的描述
- db2diag -h notes 提供用法说明和限制
- db2diag -h examples 提供一小组示例以帮助您入门
- db2diag -h tutorial 提供所有可用选项的示例
- db2diag -h all 提供最完整的选项列表

8.3.4 db2pd

db2pd 是用于诊断和监视各种 DB2 数据库活动以及故障排除的监控工具。它是从 DB2 V8.2 开始随 DB2 引擎发布的一个独立的实用程序，其外观和功能类似于 Informix onstat 实用程序(其实就是 IBM 收购 Informix 后，DB2 从 Informix 数据库那里借鉴过来的)。db2pd

是从命令行以一种可选的交互模式执行的。该实用程序运行得非常快，因为它不需要获取任何锁，并且在引擎资源以外运行(这意味着它甚至能在一个挂起的引擎上工作)。db2pd 功能非常强大，下面我们举一个使用 db2pd 生成堆栈跟踪的例子。

可对 Windows 操作系统使用 db2pd -stack all 命令(对 UNIX 操作系统使用时为-stack 命令)来对当前数据库分区中的所有进程生成堆栈跟踪。如果您怀疑某个进程或线程正在循环或正被挂起，那么可能要反复使用此命令。

可以通过发出命令 db2pd -stack <eduid>来获取特定引擎可调度单元(EDU)的当前调用堆栈。例如：

```
db2pd -stack 137
```

正在尝试转储 eduid 137 的堆栈跟踪。

如果需要所有 DB2 进程的调用堆栈，那么请使用 db2pd -stack all 命令尝试转储实例的所有堆栈跟踪，生成文件在诊断路径下。例如，在 Windows 操作系统上：

```
db2pd -stack all
```

db2pd 功能非常强大，此处我们仅仅介绍一下，关于这个工具的详细使用，请参见《深入解析 DB2-DB2 高级管理、内部体系结构和诊断案例》一书。

8.3.5　DB2 内部返回码

有两种类型的内部返回码：ZRC 值和 ECF 值。这些返回码一般仅在供 IBM 软件支持机构使用的诊断工具中可视。例如，它们出现在 DB2 跟踪输出和 db2diag.log 文件中。

ZRC 和 ECF 值基本上具有相同的作用，但格式上稍有不同。每个 ZRC 值都具有以下特征：类名、组件、原因码、相关联的 SQLCODE、SQLCA 消息标记、描述。ECF 值由以下部分组成：集名、产品标识、组件、描述。

ZRC 和 ECF 值通常为负数，并用于表示错误状况。ZRC 值根据它们表示的错误类型进行分组。这些分组称为“类”。例如，具有以“SQLZ_RC_MEMHEP”开头的名称的 ZRC 值通常是与内存不足相关的错误。相似地，ECF 值被分组为“集”。

以下是包含 ZRC 值的 db2diag.log 条目的示例：

```
2006-02-13-14.34.35.965000-300    I17502H435       级别：错误
PID：940                     TID：660          PROC：db2syscs.exe
实例：DB2                    节点：000        数据库：SAMPLE
APPHDL：0-1433                APPID：*LOCAL.DB2.050120082811
函数：DB2 UDB，数据包含，sqlpsize，探测点：20
返回码：ZRC=0x860F000A=-2045837302=SQLO_FNEX“找不到文件。”DIA8411C 找不到文
件 ""。
```

可使用 db2diag 命令获取有关此 ZRC 值的完整详细信息，例如：

```
c:\>db2diag -rc 0x860F000A
输入 ZRC 字符串“0x860F000A”被解析为 0x860F000A(-2045837302)。---Windows
要映射的 ZRC 值：0x860F000A(-2045837302)
        V7 等同 ZRC 值：0xFFFFE60A(-6646)
ZRC 类：        关键介质错误(类索引：6)
组件：         SQLO；操作系统服务(组件索引：15)
原因码：         10(0x000A)
标识：  SQLO FNEX
        SQLO_MOD_NOT_FOUND
标识(不带组件)：        SQLZ RC FNEX
描述：         未找到文件。
相关信息：        Sqlcode -980
SQL0980C  发生磁盘错误。无法处理后续的 SQL 语句。
sqlca 标记数：0        对话框消息号：8411
```

如果发出命令 db2diag -rc -2045837302 或 db2diag -rc SQLO_FNEX，将返回相同信息。

ECF 返回码的输出示例如下：

```
c:\>db2diag -rc 0x90000076
输入 ECF 字符串“0x90000076”被解析为 0x90000076(-1879048074)
要映射的 ECF 值：0x90000076(-1879048074)
ECF 集：        setecf(集索引：1)
产品：        DB2Common
组件：         OSSe
代码：         118(0x0076)
标识：         ECF_LIB_CANNOT_LOAD
描述：         不能装入指定的库。
```

db2diag 命令输出中最有价值的故障诊断信息是描述和相关信息(仅对于 ZRC 返回码)。要获取 ZRC 或 ECF 值的完整列表，请分别使用命令 db2diag -rc zrc 和 db2diag -rc ecf。

8.4 故障诊断分析流程

8.4.1 故障诊断流程

图 8-4 列出了当数据库出现故障时，故障诊断检查列表。

DB2故障诊断检查列表

- 详细的故障描述：问题症状是什么？问题是在哪里发生的？问题何时发生的？发生问题的条件是什么？问题是否可以再现？
- 获取错误信息SQLCODE/SQLSTATE/Reason Code/操作系统错误日志判断是否存在硬件故障
- 结合操作系统判断是否有硬件故障和操作系统层面的软件错误
- 读取db2diag.log文件，dump文件,trap文件和db2cos脚本输出文件
- 操作系统版本补丁；利用db2ls和db2level判断DB2产品补丁信息
- 使用db2support收集故障信息；使用db2diag解释内部返回码，使用db2诊断工具来进一步定位问题。
- 结合操作系统、应用和数据库综合判断；分析数据库配置参数，监控数据库运行情况。
- 结合厂商判断是否DB2数据库bug。

图 8-4　DB2 故障诊断检查列表

我们可以看到，当系统出现故障时，为了准确地分析问题，第一步要做的就是完整地描述问题。如果没有问题描述，您就不知道从什么地方开始调查造成问题的原因。此步骤包括询问自己如下基本问题：

- 症状是什么？
- 问题是在哪里发生的？
- 问题是在何时发生的？
- 发生问题的条件是什么？
- 问题是否可以再现？

通过回答上述及其他问题，就得到了对大多数问题的准确描述，并且也是找出问题解决方案的最好办法。

症状是什么？

开始描述问题时，最明显的问题是“发生了什么问题？”。这看起来像一个很直观的问题；但是，它可以分为若干其他问题，从而更好地描述该问题。这些问题包括：

- 是什么人或什么工具报告该问题的？
- 错误代码和错误消息是什么？
- 怎么失败的？例如循环、中止、崩溃、性能下降或结果错误。
- 对业务有什么影响？

问题是在哪里发生的？

确定问题发生的位置并不总是那么容易，但它是解决问题的最重要步骤之一。报告组件和失败组件之间可能存在多层技术。网络、磁盘和驱动程序只是调查问题时要考虑的几个部分。

- 是在特定平台上还是在多个平台上都发生了该问题？
- 是否支持当前环境和配置？
- 应用程序是在数据库服务器本地运行还是在远程服务器上运行？
- 是否涉及网关？
- 数据库是位于各个磁盘上，还是位于 RAID 磁盘阵列上？

这些类型的问题可帮助您隔离问题层，并且是确定问题来源所必需的。记住，不能只因为某层报告问题而总是断定那就是问题根源所在。

标识发生问题的位置时应了解发生问题的环境。总是应该花一些时间来完整描述问题环境，包括操作系统、操作系统版本、所有相应软件及版本和硬件信息。确认您正在配置受支持的环境中运行，这是因为许多问题会解释为发现若干软件级别不能在一起运行，或者未在一起经过完整测试。

问题是在何时发生的？

问题分析中的另一个必需步骤是创建导致故障的事件的详细时间线，对于从前发生的那些案例而言尤其如此。可以通过回溯工作过程很轻松地完成此任务：从报告错误时开始(尽可能精确，甚至精确到毫秒)，然后通过可用日志和信息回溯工作过程。通常您只需要进行观察，直到您在任何诊断日志中发现的第一个值得怀疑的事件为止，但是，这并不总是那么容易，通常需要您有实践经验。如果同时存在多层技术，每层都有自己的诊断信息，那么要知道停止的时间就特别困难。

- 问题是否仅在日间或夜间的特定时间发生？
- 问题多久发生一次？
- 导致问题的事件的发生顺序是什么？
- 问题是否发生在环境改变(如升级现有软件或硬件或者安装新的软件或硬件)之后？

回答这类问题可帮助您创建事件的详细时间线，并且为您提供用来进行调查的参考框架。

发生问题的条件是什么？

要完整地描述问题，知道发生问题时还有什么别的软件在运行是非常重要的。如果问题是在特定环境或特定条件下发生的，那么这可能是找出问题原因的关键线索。

- 问题是否总是在执行同一任务时发生？
- 事件是否要按一定顺序发生，问题才会再现？

- 是否存在其他应用程序同一时间失败？

回答这些类型的问题可帮助您说明发生问题的环境，并且能够将所有从属项关联起来。记住，不能认为只因为同一时间发生了多个问题就表示它们总是相关的。

问题是否可以再现？

从问题描述和调查角度来看，“理想”的问题是可以再现的。对于可再现的问题来说，您几乎总是可以将它们与提供的一大堆工具或过程配合使用，以帮助进行调查。因此，可再现问题通常比较容易调试和解决。

但是，可再现问题也有缺点：如果该问题对企业有很严重的影响，那么您可能不想让它再现。在这种情况下，最好尽可能在测试或开发环境中再现该问题。

- 能否在测试机器上再现问题？
- 是否多个用户或应用程序都遇到同一类型的问题？
- 能否通过运行单个命令、一组命令、特定应用程序或独立应用程序来再现问题？
- 能否通过从 DB2 命令行输入等价命令/查询来再现问题？

因为在调查时测试或开发环境有更好的灵活性，也更便于控制，所以最好在测试或开发环境中再现容易发生的单个问题。

8.4.2　结合系统事件判断

每个操作系统都有一组自己的诊断文件，用于跟踪活动和故障。最常见的(通常也是最有用的)诊断文件是错误报告或事件日志。

通常会忽略系统消息和错误日志。如果在问题定义和调查的初始阶段花时间执行一个简单任务，那么就可以在解决问题时节省几小时、几天甚至几星期的时间。该任务将会比较不同日志中的各个条目，并且记录看起来与时间和各个条目引用的资源相关的所有内容。

虽然并非总是与问题诊断有关，但许多情况下系统日志中会提供最好的线索。如果可以将报告的系统问题与 DB2 错误关联起来分析，那么通常就能确定直接导致 DB2 症状的原因。很明显的示例包括磁盘错误、网络错误和硬件错误。并不那么明显的示例包括在不同机器上报告的错误，机器不同会影响连接时间或认证。

可以调查系统日志以评估稳定性，特别是在全新的系统上报告问题时尤其如此。在常用应用程序中间歇发生陷阱可能表示存在底层硬件问题。

以下是系统日志提供的一些其他信息：

- 重要事件，如重新启动系统的时间
- 系统上发生 DB2 陷阱(即失败的其他软件中的错误、陷阱或异常)的时间

- 内核应急启动、文件系统空间不足和交换空间不足错误(可能导致系统无法创建或派生新进程)

系统日志可帮助您在 db2diag.log 中排除作为考虑因素的崩溃条目。如果在 DB2 管理通知或 DB2 诊断日志中发现崩溃恢复但先前没有任何错误，那么 DB2 崩溃恢复可能是系统关闭造成的。以下是一些操作系统上的常见系统日志：

- AIX：使用/usr/bin/errpt -a 命令
- Solaris：/var/adm/messages*文件或/usr/bin/dmesg 命令
- Linux：/var/log/messages*文件或/bin/dmesg 命令
- HP-UX：/var/adm/syslog/syslog.log 文件或/usr/bin/dmesg 命令
- Windows：使用系统日志文件、安全性日志文件、应用程序事件日志文件以及 windir\drwtsn32.log 文件(其中，windir 是 Windows 安装目录)

8.4.3 结合系统运行状况诊断

在诊断与内存、交换空间、CPU、磁盘存储器和其他资源有关的一些问题时，需要完整地了解给定操作系统管理这些资源的方式。至少在确定与资源有关的问题时需要知道对于每个用户而言，该资源存在的限制及限制程度(相关限制通常用于 DB2 实例所有者的用户标识)。例如在 AIX 上我们可以在/etc/security/limits 文件中，设置 DB2 实例所有者 db2inst1 在内存 RSS、CPU、data、core、stack、文件系统 fsize 和 nofiles 等的资源使用限制。

以下是需要获取的一些重要配置信息：

- 操作系统补丁级别、已安装软件和升级历史
- CPU 数目
- RAM 量
- 交换和文件高速缓存设置
- 用户数据和文件资源限制及每个用户的进程极限
- IPC 资源限制(消息队列、共享内存段和信号量)
- 磁盘存储器类型
- DB2 是否争用资源
- 认证在何处进行

大多数平台有直接的命令可用来检索资源信息。但是，您很少需要手动获取该信息，原因是 db2support 实用程序会收集此数据及更多其他信息。当指定了-s 和-m 选项时，db2support 生成的 detailed_system_info.html 文件包含用来收集此信息的许多操作系统命令的语法。

8.5　本章小结

本章我们讲解了关于故障诊断的一些知识，其实在一个复杂的应用环境中，故障诊断特别复杂。而且 DB2 相对来说提供的故障诊断日志稍显封闭(相对于 Oracle)，所以很多问题的诊断必须建立在对 DB2 内部体系结构非常了解的基础上才能作出准确的判断。关于 DB2 高级诊断的知识大家可以参考《深入解析 DB2》一书。



DB2 性能监控

在系统出现这样或那样的性能问题时，你怎么知道什么地方才是性能调整努力的焦点呢？这就需要对数据库进行性能监控。

DBA 在动手做性能调整时一定要制订明确详细的计划，在头脑里一定要有个现实的可测量的目标。否则就会成为一个无意义的练习。在改进数据库性能时，必须首先清楚哪里会是性能的瓶颈并有相应的对策，这就是 DB2 性能监控工具的用武之地。

本章主要讲解如下内容：

- 监控工具概述
- 快照监视器
- 快照监控案例讲解
- db2pd 及监控案例
- 事件监视器及监控案例
- db2mtrk 及监控案例
- 活动监视器
- 性能监控总结

9.1 监控工具概述

DB2 数据库为我们提供了很多监控工具，如快照监视器、事件监视器、db2pd 工具、db2mtrk、Activity Monitor 等。要调整数据库配置参数，就必须监控数据库以获得有关锁定、连接、缓冲池使用、表空间使用和内存使用等方面的信息。而要收集资源使用的详尽信息，就必须使用 DB2 数据库监控工具。可以在 DB2 客户机或 DB2 服务器上执行监控功能。要调用监控工具，可使用 CLP 命令、图形性能监控界面或调用监控用的 API 接口。

Snapshot Monitor(快照监视器)提供在特定时刻有关数据库活动的信息。它是数据库活动当前状态的图像。当拍下快照时，返回给用户的数据量由监视器开关来决定。这些开关可以在 DBM 配置文件中或在会话级别上设置。

Event Monitor(事件监视器)记录 DB2 数据库在一段时间内的数据库活动。它允许用户收集一段时间内的信息，包括死锁、连接、SQL 语句。

监视功能的性能影响基于对被监视事件的频率和对每个事件捕获的数据量。快照监视器和事件监视器可以根据监视器定义由 DB2 立即调用。DB2 性能监控工具负责捕获和返回系统信息：快照监视器或一个或者更多个事件监视器或 db2pd。快照监视器会让您获得在一个指定的时间点上的状态映像；事件监视器在指定时间段内获取监视器数据并将它们记入文件(命名管道)或表。

db2pd 工具是 DB2 V8.2 以后提供的一种非常强大的监控工具。用于收集 DB2 实例和数据库的统计信息。与 Informix Dynamic Server 的 onstat 工具类似，db2pd 提供了 20 多个选项用于显示关于数据库事务、表空间、表统计信息、动态 SQL、数据库配置和其他很多数据库细节的信息。db2pd 命令可以检索多个领域的信息，并把结果保存到文件中。也可以在特定时期内多次调用该工具，以帮助您了解随着时间的变化数据库中的变动情况。该工具可用于故障诊断、问题确定、数据库监控、性能调优和帮助应用程序的开发设计。

数据库性能监控工具使用这些监控要素的一些组合来获取监视数据，并为每一个要素的数据存储提供了多种选择。对于快照和事件监视器，您可以选择将所有收集的数据存放在文件或数据库表中，只需通过屏幕察看或者使用一个定制的程序去处理即可。数据库系统监视器通过一些使用自描述的数据流来将监视数据返回到客户端应用程序。使用快照监视应用程序，您可以调用适当的 API 来获得快照信息，然后处理返回的数据流。使用事件监视应用程序，须要事先准备从文件或表来接收数据，并激活相应的事件监视器，然后像刚才接收数据那样来处理数据流。快照监视器被设计用于收集那些在它控制下的某一特定时间点的 DB2 实例和一些数据库的状态信息。快照对于确定一个数据库系统的实时状态是非常有用的。采用固定的时间间隔，它们能够提供一些用于观察性能趋势走向和辨认潜在问题范围的信息。快照监视通过在命令行处理器(CLP)中执行 GET SNAPSHOT 命令。虽然快照监视器信息有助于诊断问题范围，但收集数据经常会引起额外的处理负担。例如，为了计算执行 SQL 语句总的时间花费，DB2 数据库管理器必须在 SQL 语句执行前后去调用操作系统级命令用来获得时间戳信息。这些系统级调用的成本可能是昂贵的。其他副作用是增加内存的使用：快照监视器数据是收集和存放在内存(DBM 的 MON_HEAP_SZ 参数)中而不是存放在某些特定的表或者外部文件中。为了有助于减少收集快照监视器数据的过载需求的数量，DB2 推荐使用控制被收集性能的数量和类型的快照监视器开关。像其他基本开关一样，每个快照监视器都有开和关两种状态。当快照监视器开关打开时，在这个开

关控制之下的一些监视器要素的信息被收集起来。相反也是(注意：一定数量的监视信息是不受这些开关的控制的，因此，这些信息不管那些开关被设置成什么状态总会被收集的。例如 timestamp 监视信息)。

9.2　快照监视器

9.2.1　快照监视器概述

在数据库管理器级别，可以通过数据库管理器配置参数来设置监视器开关。要查看所有用于监视器开关设置的设置选项，请使用 get dbm cfg | grep DFT_MON(在 Windows 执行命令 get dbm cfg |find /i “dft_mon”)。要启用或禁用数据库管理器级别的监视器开关设置，请使用 UPDATE DBM CFG 命令，并指定要更改的个别监视器开关。例如，以下命令关闭了 DFT_MON_TIMESTAMP 监视器，来终止时间戳记监视器数据的收集：

```
db2 update dbm cfg using DFT_MON_TIMESTAMP off
```

每个连接至数据库的应用程序(会话)都有其自己的监视器开关集，这些监视器开关与数据库管理器和其他应用程序(会话)无关。应用程序(会话)在连接至数据库时，从数据库管理器上继承它们的监视器开关设置。要查看会话的所有监视器开关设置的设置选项，请使用 GET MONITOR SWITCHES 命令。您可以使用 UPDATE MONITOR SWITCHES 命令来更改会话的监视器开关设置。例如，下面的命令打开 LOCK 监视器开关，从而启用 SNAPSHOT_LOCK 快照表函数所使用的监视器元素的收集：

```
db2 update monitor switches using LOCK on
```

Snapshot Monitor(快照监视器)以计算器的形式提供累计信息。快照信息由一种特殊的数据结构提供，这种数据结构可以通过应用程序发出快照来检查。由监视数据结构返回的数据根据表 9-1 中所定义的开关来设置。这些开关可以在实例(DBM 配置)级别或应用程序会话级别(UPDATE MONITOR SWITCHES)上打开或关闭。表 9-1 还包括执行快照时所提供的概要信息，以及由 Snapshot Monitor 提供的基本信息。

表 9-1　快照监视器开关参数

组　别	所提供的信息	监视器开关	DBM 参数
排序	所用堆的数目、溢出、排序性能	SORT	DFT_MON_SORT
锁定	保持锁定数目、死锁数目	LOCK	DFT_MON_LOCK
表	测量活动(读行、写行)	TABLE	DFT_MON_TABLE

(续表)

组　别	所提供的信息	监视器开关	DBM 参数
缓冲区	读和写的次数，所用时间	BUFFERPOOL	DFT_MON_BUFPOOL
工作单元	开始时间、结束时间、完成时间	UOW	DFT_MON_UOW
SQL 语句	开始时间、停止时间、语句标识	STATEMENT	DFT_MON_STMP

在 DBM(实例)配置参数中设置默认的开关值，将影响该实例中的所有数据库。而且每个与数据库相连接的 session(会话)将继承在 DBM 配置中所设置的默认开关值。

1. 查看监控开关设置

在某种程度上，由于快照监视器开关控制着当一个快照被打开时所能够收集到的信息的类型和数量，因此，您应该在开始您的监视进程之前搞清楚哪些开关是打开的而哪些是关闭的。要获得这些信息，最简单的方法就是在 CLP 中执行 GET MONITOR SWITCHES 命令。在多分区数据库环境下它的基本语法是：

```
GET  MONITOR  SWITCHES  < AT DBPARTITIONNUM  [PartitionNum] >
```

其中，PartitionNum 参数用来说明您需要获取快照监视器开关参数状态的数据库分区。

注意：

尖括号(<>)显示的参数是可选参量，而方括号([ ])里的参数是必需的。要获取和显示一个单独分区数据库的快照监视器开关的状况，可以执行 GET MONITOR SWITCHES 命令。假定均使用默认设置，结果如例 9-1 所示。

例 9-1　运行 GET MONITOR SWITCHES 命令的结果。

```
D:\>db2 get monitor switches
            监视器记录开关
数据库分区号 0 的开关列表
缓冲池活动信息        (BUFFERPOOL)  = OFF
锁定信息                    (LOCK)  = OFF
排序信息                    (SORT)  = OFF
SQL 语句信息           (STATEMENT)  = OFF
表活动信息                 (TABLE)  = OFF
获取时间戳记信息(时间戳记) = ON  2008-10-09 13:27:53.235344
工作单元信息                 (UOW)  = OFF
```

从上面可以看到，TIMESTAMP 这个快照监视器开关的状态是 ON，而其他的都是 OFF。在这个开关状态后面显示的是这个开关打开的精确日期和时间。

2. 改变开关设置

在知道了每一个可用的快照监视器开关的当前状态后，您就可以在开始监控之前去改变其中的一个或者多个开关的设置，从而获得相应的监控信息。您可以通过改变相对应的数据库管理器配置参数(该设置在实例重启后依然有效)，或者调用应用程序级 db2MonitorSwitches() API 函数或执行 UPDATE MONITOR SWITCHES 命令在会话级修改快照监视器开关的设置，可以设置的开关参见表 9-1，这个命令的基本语法是：

```
UPDATE MONITOR SWITCHES USING [[SwitchID] ON | OFF ,...]
```

其中 SwitchID 指明一个或者多个需要改变状态的快照监视器开关(该参数可以是以下其中的一部分或者是全部：BUFFERPOOL、LOCK、SORT、STATEMENT、TABLE、TIMESTAMP 和 UOW)。要将 LOCK 和 SORT 快照监视器的开关参数状态设置为 ON(会话级别)，可以执行 UPDATE MONITOR SWITCHES USING LOCK ON SORT ON 命令。

3. 获取数据

当数据库被激活或者与数据库的连接被建立时，快照显示器就会自动地开始收集监视器数据。但是，在希望能够查看被收集的数据之前，您必须选取一个快照(快照看起来就像是当时那个时间点上的监视要素的映像)。您可以通过调用 db2GetSnapshot() API 或者执行 GET SNAPSHOT 命令来得到快照。例 9-2 指明了这个命令的基本语法，Database Alias 用来说明需要做快照监视器信息的数据库别名。

这两种方法所收集的信息都存储于监视器要素中(有时被认为是数据要素)，每个要素被设计成存储指定类型的信息。下面列出的是可利用的监视器要素：

1) 计数器

用来保存活动或者事件发生次数的累计值(例如，对于一个数据库的已经执行的 SQL 语句的总次数)。计数器数值的增长贯穿监视器的生命周期；而在许多情况下，它有可能会被重置。

2) 计量值

表明一个项目的当前值(例如，当前连接到数据库的应用程序的数量)。

与计数器值不同的是，Gauges(计量)的值可以变高或者变低；它们在任一被测量点的实时值通常取决于数据库活动的级别。

3) 高水位值

表明一个指标在监视开始以后所能达到的最大值或最小值(例如，util_heap_sz 使用的

最大值)。

4) 信息要素

提供所有监视活动执行的细节信息(例如缓冲池名称、数据库名称和别名、详细路径等等)。

5) 时间戳

表明一个活动或者事件发生的日期和时间(例如第一次连接数据库建立的日期和时间)。时间戳被看成是从 1970 年 1 月 1 日开始消逝的秒和微妙的数量的值。

6) 时间要素

记录时间被花费于执行一个活动或事件的成本(例如：进行排序操作的时间花费)。时间要素的值会以从活动或事件开始所流逝的秒和微秒的数量形式来表现。一些时间要素可以被重置。

使用 GET SNAPSHOT 命令可以要求一个快照。在我们检查 Snapshot Monitor(快照监视器)的输出之前，使我们可以选择如何捕获快照信息。可以利用 CLP 界面去捕获数据库监视器快照。

拍快照时，有可能定义一个相关领域。当需要数据库监视时，通常都有具体的需要。因此，对于特定的数据库，如果有一个与并发性(锁定)行为有关的问题，那么可以规定锁定级别。如果用户关心所有访问数据库的应用程序，那么就应当规定应用程序的级别。如果 STATEMENT 开关被打开并在数据库级别上拍了快照，那么就能捕获有关表的动态 SQL 语句的活动(INSERT/ UPDATE/ SELECT/DELETE)信息。还能捕获数据库内每个表的数据库活动信息。可以提供以下级别的快照：

- DBM(实例监视器)——捕获活动实例的信息
- Database(数据库)——捕获所有数据库或一个数据库的信息
- Application(应用程序)——捕获所有应用程序或单个应用程序的信息
- Table Space(表空间)——捕获数据库内各个表空间的信息
- Table(表)——捕获数据库内各个表的信息
- Lock(锁)——捕获使用数据库的应用程序持有的各种锁的信息

例 9-2 GET SNAPSHOT 命令的句法。

```
Db2 get snapshot for dbm
Db2 get snapshot for database on dbname
Db2 get snapshot for tablespaces on dbname
Db2 get snapshot for bufferpools on dbname
Db2 get snapshot for tables on dbname
```

```
Db2 get snapshot for locks on dbname
Db2 get snapshot for applications on dbname
Db2 get snapshot for dynamic sql on dbname
```

仅仅想得到在 sample 数据库中被应用程序保持的锁定的快照信息，可以执行 GET SNAPSHOT FOR LOCKS ON sample 命令。该命令输出的监控信息类似于例 9-3 中的结果(须要注意的是，这只是一个非常简单的例子，监视器真正返回的监视数据通常要比这个大得多)。

例 9-3　数据库锁定快照。

```
数据库名称                                = SAMPLE
数据库路径                            = C:\DB2\NODE0000\SQL00001\
输入数据库别名                        = SAMPLE
挂起的锁定                    = 45
当前已连接的应用程序          = 1
当前正等待锁定的代理程序数     = 0
快照时间戳记                   = 2008-10-15 09:20:20.234487
应用程序句柄                                = 13
应用程序标识                          = *LOCAL.DB2.081015011941
序号                                  = 00001
应用程序名                            = db2taskd
CONNECT 授权标识                              = ORACLE
应用程序状态                          = 连接已完成
状态更改时间                          = 2008-10-15 09:20:20.234487
应用程序代码页                              = 1386
挂起的锁定                    = 28
总计等待时间(毫秒)           = 23322.12
应用程序句柄                                = 12
应用程序标识                          = *LOCAL.DB2.081015011940
序号                                  = 00001
应用程序名                            = db2stmm
CONNECT 授权标识                              = ORACLE
应用程序状态                          = 连接已完成
略………
```

我们可以监控同一实例控制下所有活动数据库的数据库数据、应用程序数据、缓冲池活动数据、表空间数据、表数据、锁的活动(关于所有保持锁定的锁的信息)、动态 SQL 数据(在 SQL 语句缓存中的当时关于 SQL 语句的信息)。

在后面的一些小节中，快照监视可用作寻找 DBM 和 DB 配置参数的最优设置的一种方式。

4. 重置计数器

计数器用于保存一个运行期间的具体活动或事件发生的次数的数量的累积，典型的计数开始于快照监视器开关打开或与数据库的连接被建立的时候(如果实例级别的监视器被使用，计数开始于应用程序第一次建立与该实例控制下的数据库连接的时候)。计数一旦开始，它将一直继续直到快照显示器开关被关闭或所有数据库连接被终止为止。但是，有时候也可以在没有改变一个或更多快照监视器开关状态，和没有终止或重建所有当前活动数据库连接的情况下，重置所有计数器为零。在这种情况下，可以通过执行 RESET MONITOR 命令将所有的监视器的计数器归零。这个命令的基本语法是：RESET MONITOR ALL 或者 RESET MONITOR FOR [DATABASE | DB] [DatabaseAlias]。如果您想要重置一个实例控制下的所有数据库快照监视器的计数器，可以切换到这个实例下，然后执行 RESET MONITOR ALL 命令。另一方面，如果您只是想要把与 sample 数据库相关联的快照监视器的计数器重置为 0 的话，那么可以执行 RESET MONITOR FOR DATABASE SAMPLE 命令。记住，您不能使用 RESET MONITOR 命令来有选择性地对快照监视器开关所控制的特殊的监视器组重置它们的计数器，您必须将适当的快照监视器开关关闭，然后再打开或者终止后重建数据库连接。在命令行调用快照监视器只是调用快照监视器的一种方法，并且在有些时候在命令行调用快照并不是很好的选择。下面将会简单介绍另外一些可以使用的调用快照监视器的技术。

9.2.2 利用表函数监控

DB2 数据库从 V8 版本后提供了很多表函数，利用这些表函数可以获得很多与数据库性能有关的信息，因而也就可以通过使用这些表函数代替 get snapshot 命令来收集这些监控数据。

快照表函数能够捕获的许多监视器元素都受控于监视器开关。如果快照表函数中的某些函数描述中提到特定的监视器开关，则表明受控于该开关。

DB2 的早些版本中无法使用 SQL 来捕获数据，获取快照监视器数据唯一的方式就是去执行 GET SNAPSHOT 命令或者从应用程序中调用它对应的 API 函数。而在 DB2 UDB 8.1 中，我们可以通过引用 20 多个可用的快照监视器表函数中的一个来构造查询去收集快照数据。表 9-2 列出了这些表函数以及它们所能获取的具体快照信息。更多的表函数请查询 DB2 相关文档。

在 DB2 V8 中能够使用 SQL 表函数捕获快照，这是一种明显的改进，从而可以轻松地捕获并存储快照，以便快速且灵活地进行检索。

表 9-2 部分表函数列表

快照表函数	返回的信息
SNAPSHOT_DBM	数据库管理器信息
SNAPSHOT_DATABASE	数据库信息。只有当至少有一个应用程序连接至数据库时，才会返回信息
SNAPSHOT_APPL	连接至分区上数据库的应用程序上有关锁等待的应用程序信息，包括累积计数器、状态信息和最近执行的 SQL 语句(如果设置了语句监视器开关)
SNAPSHOT_APPL_INFO	每个连接至分区上数据库的应用程序的常规应用程序标识信息
SNAPSHOT_LOCKWAIT	有关锁等待连接至分区上数据库的应用程序的应用程序信息
SNAPSHOT_STATEMENT	有关连接至分区上数据库的应用程序的语句的应用程序信息，包括最近执行的 SQL 语句(如果设置了语句监视器开关)
SNAPSHOT_TABLE	连接至数据库的应用程序所访问的每个表的表活动信息。需要表监视器开关
SNAPSHOT_LOCK	数据库级别上的锁信息，以及每个连接至数据库的应用程序在应用程序级别上的锁信息。需要锁监视器开关
SNAPSHOT_TBS	数据库级别上的表空间活动信息，每个连接至数据库的应用程序在应用程序级别上的表空间活动信息，以及连接至数据库的应用程序已访问过的每个表空间在表空间级别上的表空间活动信息。需要缓冲池监视器开关
SNAPSHOT_BP	指定数据库的缓冲池活动计数器。需要缓冲池监视器开关
SNAPSHOT_DYN_SQL	来自用于数据库的 SQL 语句高速缓存的某个时间点语句信息

快照监视器数据组织

所有的快照表函数都返回一张监视器数据表，其中的每一行代表一个正被监控的数据库对象实例，而每一列代表一个监视器元素(监视器元素代表数据库系统状态的特定属性)。

捕获监视器数据快照

要使用快照表函数捕获直接访问的快照，请完成以下步骤：

(1) 连接至数据库。它可以是您需要监控的实例中的任何数据库。要使用快照表函数发出 SQL 查询，您必须连接至数据库。例如：

```
db2 connect to sample
```

(2) 确定您需要捕获的快照类型，以及您需要监控的数据库和分区。除了收集这个信息之外，请打开任何可应用的监视器开关(通过检查表 9-2 快照表函数中的快照表函数描述信息可以确定)。

例如，如果想捕获表活动数据的快照(使用表函数 SNAPSHOT_TABLE)，那么您须要激活 TABLE 监视器开关：

```
db2 update dbm cfg using DFT_MON_TABLE on
```

(3) 使用希望的快照表函数发出查询。例如，以下是一个查询，它捕获有关当前已连接分区的 SAMPLE 数据库的表活动信息的快照：

```
db2 "select * from table(SNAPSHOT_TABLE('SAMPLE',-1)) as T"
```

快照表函数有两个输入参数：

- VARCHAR(255)，用于数据库名称。如果您输入 NULL，那么就使用当前已连接的数据库名称；

  注 1：这个参数不能应用于只返回数据库管理器信息的快照表函数(例如 SNAPSHOT_DBM)。这样的快照表函数只有一个分区号参数。

  注 2：对于快照表函数 SNAPSHOT_DATABASE、SNAPSHOT_APPL、SNAPSHOT_APPL_INFO、SNAPSHOT_LOCKWAIT、SNAPSHOT_STATEMENT 和 SNAPSHOT_BP，如果您输入 NULL 表示使用当前已连接的数据库，那么您将得到实例中所有数据库的快照信息。

- SMALLINT，用于分区号。对于分区号参数，输入一个整数(0 到 999 之间的值)以对应您需要监控的分区号。要捕获当前已连接分区的快照，请输入值 – 1 或 NULL。要捕获全局快照，请输入值 – 2。

用下面的语法可以创建一个引用非数据库管理器级表函数的查询：

```
SELECT * FROM TABLE( [FunctionName] ( [DBName], [PartitionNum]) AS [CorrelationName]
```

在这里，FunctionName 用来说明所使用快照监视器的表函数；DBName 指明需要从哪个数据库的快照监视器中收集数据；PartitionNum 用来说明需要从哪个数据库分区的快照监视器中收集数据；CorrelationName 则是查询产生的结果数据集的名称。

构造一个引用数据库管理器级的快照监视器表函数查询的语法也是一样的，只是不再使用 DBName 参数。如果想要获取一个分区数据库环境里当前分区的快照监视器数据，您可以将 PartitionNum 参数的值设置为 – 1；如果您希望获取所有分区的快照监视器数据，可

以把它设置为－2。同样，如果您想获取当前连接数据库的快照信息，可以把 DBName 参数设定成一个空值，也可以使用一对空的单引号或者用一个 CAST 操作——例如 CAST(NULL AS CHAR)。如果想要通过使用快照监视器的表函数 SNAPSHOT_LOCK 来抓取包含被应用程序相关联的当前连接的数据库的锁定数据的快照信息，可以执行下面的语句：

```
SELECT * FROM TABLE(SNAPSHOT_LOCK(CAST(NULL AS CHAR), -1) AS LOCK_INFO
```

如果我们使用 SAMPLE 数据库(先前的例子)作为当前被连接的数据库，那么执行 GET SNAPSHOT FOR LOCKS ON SAMPLE 命令后，返回的信息将会与先前的那个非常相似。

9.2.3　性能管理视图

DB2 数据库版本从 V9 后提供了很多性能管理视图(这些性能管理视图类似 Oracle 数据库中的 v$开头的动态性能视图)，使用这些管理视图可以获得与表函数和快照类似的监控数据，表 9-3 列出了部分管理视图，这些视图是 DB2 版本 9.5 的相关视图，版本 9.1 的视图略有不同，须要注意的是这些视图的模式名都是 SYSIBMADM。

同样地，这些视图中能够获得哪些监控数据很多是由监控开关的设置情况决定的。

表 9-3 部分管理视图

视　图　名	模　式　名	描　　述
APPLICATIONS	SYSIBMADM	数据库中运行的应用
APPL_PERFORMANCE	SYSIBMADM	每个应用中 rows selected 与 rows read 的比率
BP_HITRATIO	SYSIBMADM	缓冲池的命中率
BP_READ_IO	SYSIBMADM	缓冲池读的信息
BP_WRITE_IO	SYSIBMADM	缓冲池写的信息
CONTAINER_UTILIZATION	SYSIBMADM	表空间中容器的利用率信息
LOCKS_HELD	SYSIBMADM	当前获得的锁的信息
LOCKWAITS	SYSIBMADM	锁等待的信息
LOG_UTILIZATION	SYSIBMADM	日志利用率的信息
LONG_RUNNING_SQL	SYSIBMADM	执行时间最长的 SQL 语句信息
SNAPAGENT_MEMORY_POOL SNAP_GET_AGENT_MEMORY_POOL	SYSIBMADM	代理级别的内存使用情况
SNAPBP SNAP_GET_BP_V95	SYSIBMADM	缓冲池的基本信息

(续表)

视 图 名	模 式 名	描 述
SNAPDYN_SQL SNAP_GET_DYN_SQL_V95	SYSIBMADM	数据库中动态 SQL 的执行情况
SNAPLOCKWAIT SNAP_GET_LOCKWAIT	SYSIBMADM	锁等待的信息
SNAPSTMT SNAP_GET_STMT	SYSIBMADM	应用中 SQL 语句的执行情况
SNAPTAB SNAP_GET_TAB_V91	SYSIBMADM	表的信息
SNAPTAB_REORG SNAP_GET_TAB_REORG	SYSIBMADM	重组信息
SNAPTBSP SNAP_GET_TBSP_V91	SYSIBMADM	表空间信息
TBSP_UTILIZATION	SYSIBMADM	表空间利用情况
TOP_DYNAMIC_SQL	SYSIBMADM	消耗资源最多的 SQL 语句信息

表 9-3 列出了一些主要的视图，这些视图以较好的方式将获得的快照信息进行良好地组织，使获得一些性能的信息变得简单容易。

9.3 快照监视器案例

9.3.1 监控案例 1—动态 SQL 语句

例 9-4 中显示的脚本将发出一个"get snapshot for all on **dbname**"命令，该命令包括 "get snapshot for dynamic SQL on **dbname** > **snap.out**"命令的所有输出。如果您发现不会捕获很多的 SQL 语句，那么可以增加监控的时间。一条语句输出的"Dynamic SQL Snapshot Result"部分看上去如例 9-4 所示：

注意：
为了保证监控尽可能多的 SQL 语句，建议增大 DBM 配置参数 mon_heap_sz。

例 9-4 监控动态 SQL 语句。

```
Dynamic SQL Snapshot Result
```

```
    Database name                  = SAMPLE
    Database path                  = C:\DB2\NODE0000\SQL00003\
    Number of executions     = 30
    Number of compilations         = 1
    Worst preparation time(ms)     = 1624
    Best preparation time(ms)      = 1624
    Internal rows deleted          = 0
    Internal rows inserted         = 0
    Rows read                      = 1230
    Internal rows updated          = 0
    Rows written                   = 0
    Statement sorts                = 0
   Total execution time(sec.ms)  = 0.934186
    Total user cpu time(sec.ms)   = 0.000000
   Total system cpu time(sec.ms) = 0.000000
   Statement text                 = select * from sales
     ...
```

您可以看到，在输出中可以搜索一些很有用的字符串。

"Number of executions"可以帮助您找到应该调优的那些重要语句。它对于帮助计算语句的平均执行时间也很有用。

对于执行时间很长的语句，单独执行一次或许对系统消耗不多，但是累积起来的结果就会大大降低系统性能。应尽量理解应用程序是如何使用该 SQL 语句的，或许只需对应用程序逻辑稍微修改一下就可以提高性能。

看看是否可以使用参数标记(parameter marker)，以便只需为语句创建一个包，这一点也很管用。参数标记可用作动态准备的语句(生成访问计划时)中的占位符。在执行时，就可以将值提供给这些参数标记(parameter marker)，从而使语句得以运行。

有时候，为了解决某些问题，使用"grep"(UNIX)或"findstr"(Windows)对快照输出进行初步的搜索会非常方便。如果发现了什么东西，就可以通过打开快照输出来找到问题所在，以便作进一步调查。

例如，为了识别是否存在死锁，可使用如下语句：

```
   grep -n "Deadlocks detected" snap.out | grep -v "= 0" | more  -- UNIX:环境
   findstr /C:"Deadlocks detected" snap.out | findstr /V /C:"= 0" -- Windows 环境
```

例如，要搜索执行得最频繁的语句，可使用如下语句：

```
   grep -n " Number of executions" snap.out | grep -v "= 0" | sort -k 5,5rn
| more  -- UNIX 环境

   findstr /C:" Number of executions"  snap.out | findstr /V /C:"= 0" -- Windows
```

```
环境
```

注意：

在 Windows 环境下执行时，注意中英文转换，例如“Number of executions”在中文环境下应该用“执行数”代替。这里主要讲思路。

"Rows read"可帮助识别读取行数最多的状态 SQL 语句。如果读取的行数很多，通常意味着要进行表扫描。如果这个值很高，也可能表明要进行索引扫描。扫描时选择性很小，或者没有选择性，这跟表扫描一样糟糕。

您可用使用 Explain 工具来查看是否真的如此。如果有表扫描发生，那么可以对表执行一次 RUNSTATS，或者将 SQL 语句提供给 DB2Design Advisor(db2advis)，以便令其推荐一个更好的索引，以此来弥补。如果是选择性很差的索引扫描，或许需要一个更好的索引。可以试试 Design Advisor。

```
grep -n " Rows read" snap.out | grep -v "= 0" | sort -k 5,5rn
findstr /C:" Rows read" snap.out | findstr /V /C:"= 0"
```

"Total execution time" 用于将语句每次执行时间加起来，从而得到总的执行时间。我们可以用这个数字除以执行的次数，如果发现语句的平均执行时间很长，那么可能是因为表扫描和/或出现锁等待(lock-wait)的情况。索引扫描和页面获取导致的大量 I/O 活动也是一个原因。通过使用索引，通常可以避免表扫描和锁等待。锁会在提交的时候解除，因此如果提交得更频繁一些，或许可以弥补锁等的问题。

```
grep -n " Total execution time" snap.out | grep -v "= 0.0" | sort -k 5,5rn | more
findstr /C:" Total execution time" snap.out | findstr /V /C:"= 0.0" |sort /R
```

"Statement text"用于显示语句文本。如果注意到了重复的语句，这些语句除了 WHERE 子句中谓词的值有所不同以外，其他地方都是一致的，那么就可以使用参数标记，以避免重新编译语句。这样还可以使用相同的包，从而帮助避免重复的语句准备，而这种准备的消耗是比较大的。还可以将语句文本输入到 Design Advisor(db2advis)中，以便生成最优的索引。

```
grep -n " Statement text"  snap.out | more
findstr /C:"Statement text"  snap.out
```

9.3.2　监控案例 2—通过表函数监控

本案例将演示前面给出的各个步骤。在这个案例中，已有 3 个应用程序连接至 sample 数据库。两个是本地连接，另一个连接自远程客户机。一个本地应用程序已经对 sample 数

据库的 STAFF 表中的记录作了一些更新。同时，远程应用程序已经在 sample 数据库的 SALES 表中插入了一条记录。第二个本地应用程序用于执行快照监控活动。

以下是这 3 个应用程序执行的命令和语句的顺序：

设置 DFT_MON_TABLE 监视器开关。

```
db2 update dbm cfg using DFT_MON_TABLE on
```

应用程序 1(远程应用程序)：

向 SALES 表插入一条记录。

```
db2 "insert into sales values('03/20/2007','LEE','Atlantic',5)"
```

应用程序 2(本地应用程序)：

对 STAFF 表更新 12 条记录。

```
db2 "update staff set salary = salary * 1.1 where JOB = 'Clerk'"
```

捕获数据库 sample 中表信息的快照：

```
db2 connect to sample
db2 "select * from table(SNAPSHOT_TABLE('SAMPLE',-1)) as T"
```

上面查询的结果集中包含许多列，因此从命令行读取会很困难。如果您只对几个特定监视器元素感兴趣，那么可以将 select 语句限制在相关的监视器元素列。例如，以下是这样一个查询及其对应的结果集：

```
    db2 "select snapshot_timestamp, table_name, rows_written, rows_read from
table(SNAPSHOT_TABLE('SAMPLE',-1)) as T"

    SNAPSHOT TIMESTAMP          TABLE NAME        ROWS WRITTEN    ROWS READ
    -------------------------- ----------------- ------------ ------------
    2007-04-07-09.33.27.468598 SYSROUTINES                 0            4
    2007-04-07-09.33.27.468598 STAFF                      12           47
    2007-04-07-09.33.27.468598 SALES                       1            0
    2007-04-07-09.33.27.468598 SYSTABLES                   0            2
    2007-04-07-09.33.27.468598 SYSPLAN                     0            1
    2007-04-07-09.33.27.468598 SYSEVENTMONITORS            0            1
    2007-04-07-09.33.27.468598 SYSDBAUTH                   0            5
    2007-04-07-09.33.27.468598 SYSBUFFERPOOLS              0            1
    2007-04-07-09.33.27.468598 SYSTABLESPACES              0            3
    2007-04-07-09.33.27.468598 SYSVERSIONS                 0            1
    10 record(s) selected.
```

存储定期捕获的监视器快照结果可以提供大量有用信息，从而确定数据库的状态和性能趋势。要这样做，一个简单方式是对您正监控的实例数据库中的监视器数据创建一个或多个表。例如，下面这个正在创建的表将存储有关连接至实例中数据库的应用程序数的监视器数据。

```
db2 "create table instance snap(snap time timestamp NOT NULL,
local_cons bigint, rem_cons_in bigint)"
```

以下语句捕获实例中各个数据库的连接数的快照，以及时间戳记，并将这个数据插入到上面创建的 INSTANCE_SNAP 中。

```
db2 "insert into instance_snap select snapshot_timestamp, local_cons,
rem cons in from table(snapshot dbm(-1)) as snapshot dbm"
db2 "select * from instance_snap"
SNAP TIME                   LOCAL CONS          REM CONS IN
-------------------------- -------------------- --------------------
2007-04-07-09.40.49.867659                 2                  1
1 record(s) selected.
```

上面的输出结果指出有两个本地应用程序和一个远程应用程序连接到了数据库 sample 上。

有了快照表函数和性能视图，您可以使用 SQL 轻松地捕获数据库系统监视器数据的快照。将所选监视器数据集存储到 SQL 表中的这种能力使得允许许多监控应用程序存在。定期捕获并存储数据库系统的快照信息，并对特定时间段内数据库系统的使用情况和性能作统计分析。

9.3.3 编写快照监控脚本

在实际生活中，我们可能需要实时用快照监控数据库在某段时间内的活动，下面的脚本可以帮助我们在某个时间，每隔一定时间间隔来监控数据库的性能信息。

```
@echo off
REM
REM  takeAsnapshot after specified sleep period forAnumber of iterations
REM  parameters: (1) database name   ----输入数据库名称
REM              (2) file name id-----输出文件
REM              (3) interval between iterations(seconds)—监控间隔
REM              (4) maximum number of iterations-----监控次数

REM
```

```
    REM  Note: You may receive an error about the monitor heap being too small.
You may
    REM  want to set mon_heap_sz to 2048 while monitoring.---设置 mon_heap_sz
    :CHECKINPUT
    IF ""=="%4" GOTO INPUTERROR
    GOTO STARTPRG
    :INPUTERROR
    echo %0 requires 4 parameters: dbname filename_id sleep_interval iterations
    echo 举例 "getsnap.bat sample 0302 60 3"
    GOTO END
    :STARTPRG
    SET dbname=%1
    SET fileid=%2
    SET sleep_interval=%3
    SET iterations=%4
    db2 update monitor switches using bufferpool on lock on sort on statement
on table on uow on
    REM repeat the snapshot loop for the specified iterations
    SET i=1
    :SNAPLOOP
       IF %i% LSS 10 SET i2=0%i%
       IF %i% GTR 9 SET i2=%i%
       echo Starting Iteration %i2%(of %iterations%)
      DB2 -v reset monitor all
       sleep %sleep interval%
      DB2 -v get snapshot for dbm > snap%i2%_%fileid%
      DB2 -v get snapshot for all on %dbname% >> snap%i2%_%fileid%
       echo Completing Iteration %i2%(of %iterations%)
       SET /a i+=1
       IF %i% GTR %iterations% GOTO ENDLOOP
       GOTO SNAPLOOP
    :ENDLOOP
    db2 update monitor switches using bufferpool off lock off sort off statement
off table off uow off
    db2 terminate
    :END
```

上面的脚本通常用在高峰期间出现性能问题时，通过运行该脚本能够帮助我们定位性能问题所在。

9.4 db2pd 及监控案例

db2pd 是用于监视各种 DB2 数据库活动以及故障排除的监控工具。它是从 DB2 V8.2 开始随 DB2 引擎发布的一个独立的实用程序，其外观和功能类似于 Informix onstat 实用程序(其实就是 IBM 收购 Informix 后，DB2 从 Informix 数据库那里借鉴过来的)。db2pd 是从命令行以一种可选的交互模式执行的。该实用程序运行得非常快，因为它不需要获取任何锁，并且在引擎资源以外运行(这意味着它甚至能在一个挂起的引擎上工作)。通过快照监视还可以收集 db2pd 提供的很多监视器数据，但 db2pd 和快照监视的输出格式有很大不同。

下面举例来说明如何使用 db2pd 来监控数据库。

1. 监控的例子

下面这些例子说明了如何用 db2pd 工具监控您的数据库环境。

例 9-5 如果希望了解当前 DB2 的级别和当前操作系统的信息，可以输入以下命令：

```
db2pd -version -osinfo
```

效果如图 9-1 所示。

```
Command Prompt
C:\>db2pd -version -osinfo

Instance DB2 uses 32 bits and DB2 code release SQL08020
with level identifier 03010106
Informational tokens are DB2 v8.1.7.664, s040914, WR21342, FixPak 7.

Operating System Information:

OSName:    WIN32_NT
NodeName:  POONS
Version:   5.0
Release:   Service Pack 4
Machine:   x86 Family 6, model 9, stepping 5

CPU Information:
TotalCPU     OnlineCPU    ConfigCPU    Speed(MHz)   HMTDegree
1            1            1            1694         1

Physical Memory and Swap (Megabytes):
TotalMem     FreeMem      AvailMem     TotalSwap    FreeSwap
1535         917          917          2151         2175

Virtual Memory (Megabytes):
Total        Reserved     Available    Free
3686         n/a          n/a          3092

C:\>
```

图 9-1 监控例子 9-5

–version 选项显示了系统上运行的当前 DB2 的版本和级别。输入 db2level 命令可以得到相同的信息。–osinfo 选项显示 OS、CPU、物理内存和虚拟内存信息。类似的 OS 信息也可以在 DB2 启动时的 db2diag.log 中找到。这个例子也说明了获得版本信息和 OS 信息是多么简单，只需在一个 db2pd 命令中指定两个选项即可。

例 9-6 如果需要检查动态 SQL 语句的当前隔离级别，可以使用下面的命令：

```
db2pd -db sample -dynamic
```

效果如图 9-2 所示。

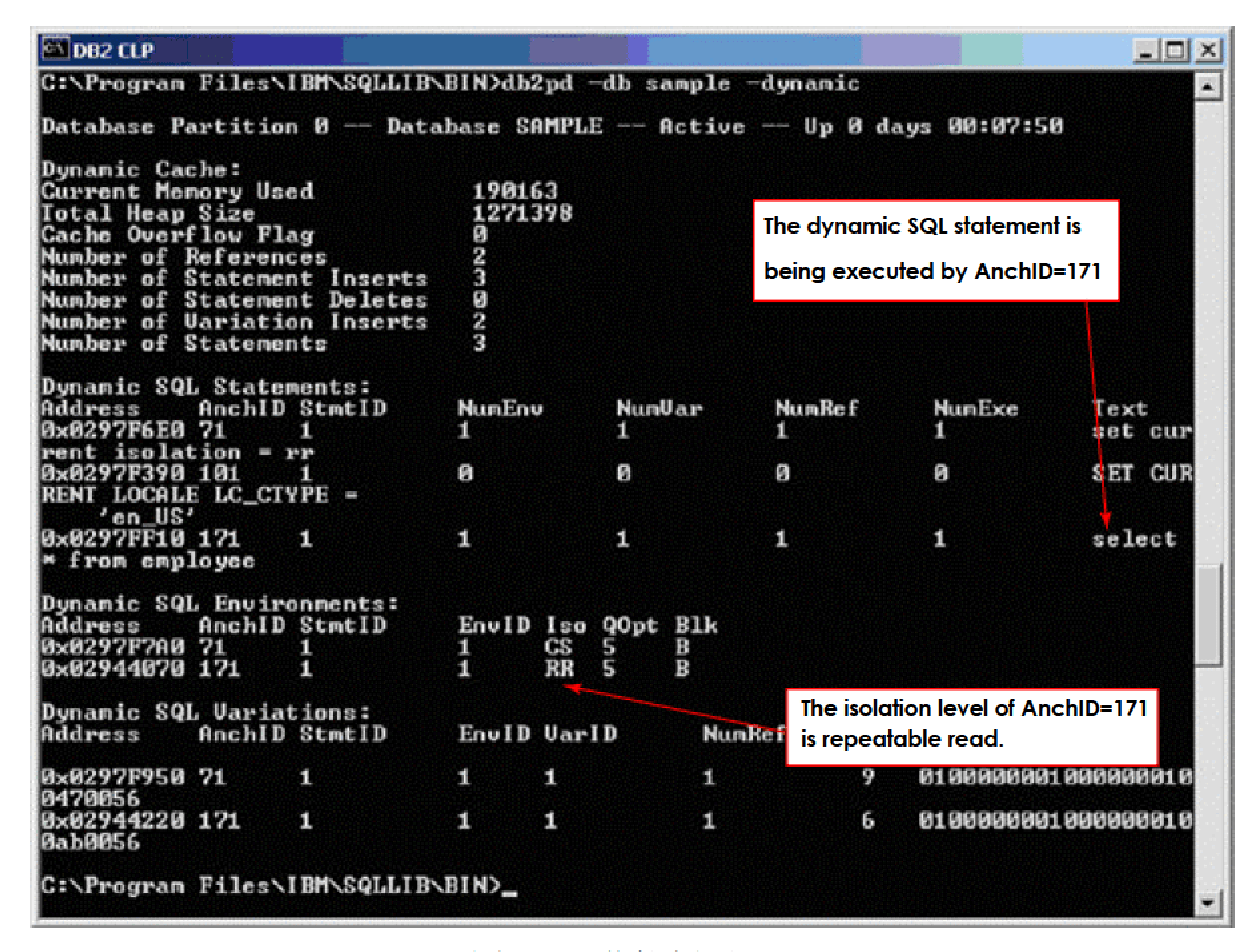


图 9-2　监控例子 9-6

图 9-2 中，在 Dynamic SQL Environments 部分可以找到执行中的动态 SQL 语句的当前隔离级别。该例中，散列的锚标识符 171(**AnchID=171**)具有最严格的隔离级别，Repeatable Read(可重复读，**RR**)。通过交叉参照 Dynamic SQL Statements，可以确定哪个具体的 SQL 语句具有 RR 隔离级别：

```
select * from employee
```

当多个 DB2 用户并发地访问一个数据库时，锁等待会导致响应变慢。锁等待是临时性的，因而难以捕捉。然而，当出现锁等待情形时，需要由数据库管理员负责确定锁等待的原因。下面通过例子演示如何使用 db2pd 和 db2pdcfg 实用程序完成该任务。

2. 用于锁监视的 db2pd 选项

图 9-3 展示了用于锁监视的 db2pd 选项。

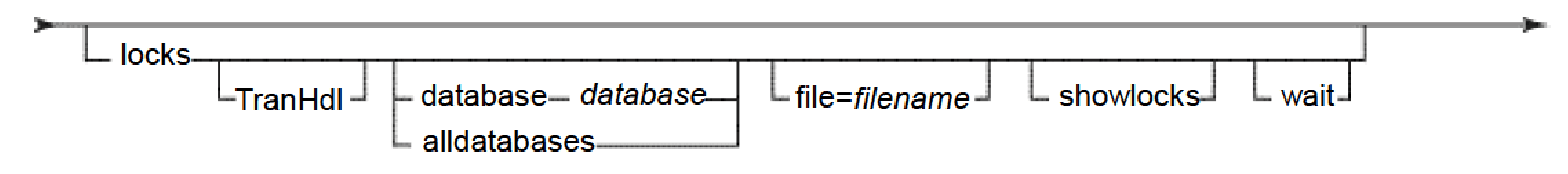


图 9-3　用于锁监视的 db2pd 选项

- TranHdl：用于指定事务句柄，以便只监视由特定事务持有的锁。
- showlocks：用于将锁名称扩展成有意义的解释。对于一个行锁，该选项显示以下信息：表空间 ID、表 ID、分区 ID、页和槽。通过使用编目视图 SYSCAT.TABLES 上的一个查询，很容易将表空间 ID 和表 ID 映射到相应的表名。

例 9-7 将表空间 ID、表 ID 映射到表模式、表名。

```
SELECT TABSCHEMA, TABNAME
FROM SYSCAT.TABLES
WHERE TBSPACEID = tbspaceid AND TABLEID = tableid
```

- wait：如果指定 wait 子选项，则 db2pd 只显示事务当前正在等待的锁，以及对等待情形负责的锁。这个子选项大大简化了锁等待分析，因为它将输出限制为参与锁等待情形的锁。
- database 和 file 选项不是特定于锁监视的，但是几乎适用于所有 db2pd 调用。database 选项将 db2pd 返回的监视器数据限制为某个数据库的监视器数据。而 file 选项则允许定义一个文件，以便将 db2pd 输出写到该文件。

3. 锁等待分析场景

用户 A 执行事务 A，以根据每个经理的薪水为他们提供 10%的奖金。

例 9-8 事务 A 执行的更新操作：

```
 UPDATE EMPLOYEE SET BONUS = SALARY * 0.1  WHERE JOB = 'MANAGER'
```

当事务 A 仍然在运行时(因为用户 A 还没有使用 COMMIT 或 ROLLBACK 终止该事务)，用户 B 执行事务 B，以将每个雇员的薪水提高 2%。

例 9-9 事务 B 执行的更新操作：

```
UPDATE EMPLOYEE SET SALARY = SALARY * 0.02
```

由于事务 B 没有完成，用户 B 请求 DBA 确定问题的原因。于是，DBA 调用 db2pd，看是否存在锁等待情形。

例 9-10 检查锁等待情形：

```
db2pd -db sample -locks wait showlocks
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:33:05
Locks:
Address    TranHdl    Lockname                  Type       Mode Sts Owner      Dur
0x050A0240 6      02000600050040010000000052 Row      ..X  W   2          1
0x050A0DB0 2      02000600050040010000000052 Row      ..X  G   2          1
```

```
HoldCount  Att  ReleaseFlg
0  0x00 0x40000000   TbspaceID 2  TableID 6  PartitionID 0 Page 320 Slot 5
0  0x00 0x40000000   TbspaceID 2  TableID 6  PartitionID 0 Page 320 Slot 5
```

db2pd 报告 ID 为 2 的表空间中一个 ID 为 6 的表上有一个行锁存在锁等待情形。通过检查 SYSCAT.TABLES，DBA 断定表 EMPLOYEE 上的确存在锁等待。

例 9-11　确定锁等待情形所涉及的表：

```
 SELECT TABSCHEMA, TABNAME FROM SYSCAT.TABLES
WHERE TBSPACEID = 2 AND TABLEID = 6
TABSCHEMA                               TABNAME
--------------------------------------------------------------------------------
ORACLE                              EMPLOYEE
  1 record(s) selected.
```

对于事务 2(列 TranHdl)，db2pd -locks 输出的 status 列(Sts)显示一个“G”。G 代表“granted”，意思就是事务句柄为 2 的事务拥有行锁。此外，列 Mode 表明，事务 2 持有的是一个 X 锁。等待的事务(列 Sts 中显示“W”(“wait”)的事务)是句柄为 6 的事务。该事务正在与事务 2 请求同一个行上的 X 锁。通过查看 Owner 列(显示事务 2 是锁的所有者)和比较 Lockname(对于 db2pd -locks 中的两个条目是相同的)，可以看到这一点。

接下来，DBA 将事务句柄映射到应用程序。这可以使用另一个 db2pd 选项-transactions 来完成：

例 9-12　将事务句柄映射到应用程序：

```
db2pd -db sample -transactions
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:34:47
Transactions:
Address    AppHandl [nod-index] TranHdl  Locks   State   Tflag     Tflag2
0x05141880 30     [000-00030]   2       9        WRITE  0x00000000 0x00000
0x05144880 34     [000-00034]   6       5        WRITE  0x00000000 0x00000
```

这个 db2pd 调用的输出表明，事务 2(列 TranHdl)是由应用程序 30(列 AppHandl)执行的，而事务 6 是由应用程序 34 执行的。这两个事务都正在对数据库执行写更改(列 State = WRITE)。所以 DBA 现在知道，应用程序 30 正持有应用程序 34 所等待的锁。

要获得关于锁等待情形涉及的应用程序的更多信息，可使用-agents 选项调用 db2pd。该选项打印代表应用程序运行的代理的信息。注意，-agents 是一个实例级选项，这意味着不需要指定一个数据库(实际上，当指定一个数据库时，db2pd 打印出一条警告，并忽略 database 选项)。

例 9-13　获得关于应用程序和相应代理的信息：

```
db2pd -agents
Database Partition 0 -- Active -- Up 3 days 08:35:42
Agents:
Current agents:      2
Idle agents:         0
Active coord agents: 2
Active agents total: 2
Pooled coord agents: 0
Pooled agents total: 0
Address    AppHandl [nod-index] AgentTid  Priority  Type    State
0x04449BC0 34       [000-00034] 3392      0         Coord   Inst-Active
0x04449240 30       [000-00030] 2576      0         Coord   Inst-Active
ClientPid  Userid   ClientNm Rowsread  Rowswrtn  LkTmOt DBName
3916       USER_B   db2bp.ex  43        43        NotSet SAMPLE
2524       USER_A   db2bp.ex  153       14        NotSet SAMPLE
```

在 db2pd –agents 的输出中，DBA 可以看到使用应用程序 30 和 34 的用户的 ID(列 Userid)：应用程序 30 是由 USER_A 执行的，而应用程序 34 是由 USER_B 执行的。只有当每个用户都有一个单独的数据库授权 ID 时，才可能出现那样的应用程序与用户 ID 之间的映射。通常，这对于在应用服务器上运行的应用程序是不可能的，因为这些应用程序使用连接池，连接不是个人化的。

关于每个应用程序的更多信息则由 db2pd 选项-applications 提供。

例 9-14 获得关于应用程序的更多信息：

```
db2pd -db sample -applications
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:36:14
Applications:
Address    AppHandl [nod-index] NumAgents  CoorTid   Status
0x04AF8080 34       [000-00024] 1          3940      Lock-wait
0x03841960 30       [000-00020] 1          2548      UOW-Waiting
C-AnchID C-StmtUID  L-AnchID L-StmtUID  Appid
195      1         0        0         *LOCAL.DB2.061122195637
0        0         60       1         *LOCAL.DB2.061122195609
```

Status 列确认了 DBA 已经知道的一些东西：应用程序 34 处于锁等待状态。但是这并不新鲜，于是 DBA 将注意力集中在列 C-AnchID/C-StmtUID 和 L-AnchID/L-StmtUID 上。“C”代表当前(current)，“L”代表最近(last)的锚 ID/语句 UID。这些 ID 可用于标识应用程序最近执行的 SQL 语句和应用程序当前执行的语句。为此，可以用-dynamic 选项调用 db2pd。该选项显示数据库动态语句缓存的内容。

例 9-15　检查动态语句缓存的内容：

```
    db2pd -db sample -dynamic
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:37:39
Dynamic Cache:
Current Memory Used           187188
Total Heap Size              1271398
Cache Overflow Flag           0
Number of References          2
Number of Statement Inserts   3
Number of Statement Deletes   0
Number of Variation Inserts   2
Number of Statements          3
Dynamic SQL Statements:
Address    AnchID StmtUID   NumEnv    NumVar    NumRef    NumExe
0x056CEBD0  60     1         1         1         1         1
0x056CE850 180     1         0         0         0         0
0x056CFEA0 195     1         1         1         1         1
Text
UPDATE EMPLOYEE SET BONUS = SALARY * 0.1 WHERE JOB = 'MANAGER'
SET CURRENT LOCALE LC_CTYPE = 'de_DE'
UPDATE EMPLOYEE SET SALARY = SALARY * 0.02
Dynamic SQL Environments:
Address    AnchID StmtUID   EnvID Iso QOpt Blk
0x056CECD0  60     1         1     CS  5    B
0x056D30A0 195     1         1     CS  5    B
Dynamic SQL Variations:
Address    AnchID StmtUID   EnvID VarID     NumRef    Typ
0x056CEEB0  60     1         1     1         1         4
0x056D3220 195     1         1     1         1         4
Lockname
010000000100000001003C0056
0100000001000000100C30056
```

-applications 输出与-dynamic 输出之间的映射很简单：

应用程序 34(处于锁等待状态)当前正在执行当前锚 ID195 和当前语句 ID1 所标识的 SQL 语句。在 db2pd -dynamic 输出的 Dynamic SQL Statements 部分中，那些 ID 可以映射到以下 SQL 语句。

例 9-16　应用程序 34 执行的 SQL 语句：

```
UPDATE EMPLOYEE SET SALARY = SALARY * 0.02
```

持有锁的应用程序 30 最近执行的 SQL 语句是最近锚 ID60 和最近语句 ID1 所标识的

SQL 语句。那些 ID 可以映射到以下 SQL 语句：

例 9-17 应用程序 30 执行的 SQL 语句：

```
UPDATE EMPLOYEE SET BONUS = SALARY * 0.1 WHERE JOB = 'MANAGER'
```

注意，db2pd -dynamic 输出包含另一个通常难以发现的有趣信息：Dynamic SQL Environments 部分的列 Iso 显示了被执行的动态 SQL 语句的隔离级别(UR = Uncommitted Read，CS = Cursor Stability，RS = Read Stability，RR = Repeatable Read)。

我们来总结一下 DBA 就用户 B 的应用程序被挂起的原因有什么发现：

- 挂起是由表 EMPLOYEE 上一个独占式的行锁导致的。
- 持有锁的事务属于用户 A 执行的一个应用程序。而用户 B 的事务正在等待那个锁。
- 两条有冲突的语句都是表 EMPLOYEE 上的 UPDATE 语句。

有了这些信息，DBA 就可以开始采取一些必要的步骤来解决锁等待状况，例如建议用户 A 终止事务，或者强制关闭用户 A 的应用程序。此外，可以采取措施避免将来出现那样的状况，例如检查是否创建了最合理的索引，使之通过索引扫描快速提交交易并释放锁。

在这个示例场景中，db2pd 被连续执行数次，每次使用一个单独的选项。现实中不会出现这样的情况。相反，db2pd 只被调用一次，调用时同时使用前面介绍的所有选项。

例 9-18 分析锁等待情形所需的带有所有选项的单个 db2pd 调用：

```
    db2pd -db sample -locks wait showlocks -transactions -agents -applications
-dynamic     -file db2pd.out -repeat 15 40
```

产生的输出由针对每个选项的输出组成，各部分输出之间的顺序与各选项在 db2pd 调用中的顺序一致。而且，请注意 db2pd 调用最后的 2 个附加选项：

- -file 表明 db2pd 输出应该被写到一个文件。在示例调用中，输出被写到文件 db2pd.out 中。
- -repeat 表明 db2pd 应该每隔 15 秒执行一次，共执行 40 次(即每隔 15 秒执行一次，共执行 10 分钟)。每次执行的输出被附加到-file 选项指定的文件后面。

-file 和-repeat 选项对于在一段时间内监视数据库活动比较有用。对于锁等待分析，这两个选项可以帮助捕捉只存在一小段时间的锁等待情形。例如，如果数据库参数 LOCKWAIT 被设置为 20 秒，一个等待锁的事务在过了 20 秒的等待时间后被回滚。为了捕捉那样的锁等待情形，db2pd 的时间间隔必须设置为比 20 秒更短的时间间隔，例如例子中的 15 秒。

4. 捕捉罕见的锁超时

有时候，锁等待会导致锁超时，而锁超时又会导致事务被回滚。锁等待导致锁超时所

需的时间段由数据库配置参数 LOCKTIMEOUT 指定。锁超时分析最大的问题是，不知道下一次的锁超时何时发生。为了捕捉死锁，可以创建一个死锁事件监视器。每当出现死锁时，这个死锁事件监视器便写一个条目。但是，对于锁超时就没有类似的事件监视器。所以到 DB2 V9 为止，捕捉锁超时的唯一方法还是连续的 db2pd 或快照监视(对于 db2pd，和前面解释的一样，-file 和-repeat 选项可用于连续的锁监视)。

DB2 V9 包含了一种新的机制，用于在数据库出现故障或发生事件时收集监视器数据—— db2cos 脚本。为了捕捉锁超时事件，可以配置数据库，使之每当出现锁超时启动 db2cos 脚本。在 db2cos 脚本中，和前面讨论的一样，可以以相同的选项调用 db2pd。我们来看一个示例场景，该场景演示了如何用 db2cos 脚本捕捉锁超时。

对于这个场景，假设 DBA 将数据库锁超时值设为 10 秒。

例 9-19　更新锁超时设置：

```
UPDATE DB CFG FOR SAMPLE USING LOCKTIMEOUT 10
```

为了每当出现锁超时时启动 db2cos 脚本，DBA 调用 db2pdcfg 实用程序，如下所示。

例 9-20　使用 db2pdcfg 配置 db2cos 脚本的调用：

```
db2pdcfg -catch locktimeout count=1
```

-catch 选项指定应该自动导致调用 db2cos 脚本的故障或事件。对于锁超时事件，可以指定字符串 locktimeout。或者，可以指定与锁超时相应的 SQL 错误码和原因码。

例 9-21　用于捕捉锁超时的另一种 db2pdcfg 调用：

```
db2pdcfg -catch 911,68 count=1
```

除了一些字符串值和 SQL 代码之外，db2pdcfg 还接受内部 DB2 错误码。所以，用这种方式可以捕捉很多数据库故障和事件。锁超时事件只是使用 db2pdcfg 和 db2cos 的一种情况。

如果 count 子选项的值为 1，则表明当出现锁超时事件时应该执行 db2cos 脚本。

db2pdcfg 通过以下输出确认错误捕捉的设置。

例 9-22　db2pdcfg 对错误捕捉设置的确认：

```
Error Catch #1
Sqlcode:         911
ReasonCode:      68
ZRC:           -2146435004
ECF:             0
Component ID:    0
LockName:        Not Set
```

```
LockType:        Not Set
Current Count:   0
Max Count:       1
Bitmap:         0x4A1
    Action:         Error code catch flag enabled
Action:         Execute sqllib/db2cos callout script

Action:         Produce stack trace in db2diag.log
```

db2diag.log 报告中也包括错误捕捉设置。可以使用 db2diag 实用程序(用于检查 db2diag.log 内容的一个有用的实用程序)过滤 db2diag.log 文件，而不必在一个文本编辑器中打开它。

例 9-23 在 db2diag.log 中确认错误捕捉设置：

```
db2diag -g funcname:=pdErrorCatch
2006-12-18-13.37.25.177000+060 I727480H285       LEVEL: Event
PID     : 4648                 TID  : 3948       PROC : db2syscs.exe
INSTANCE:DB2                  NODE : 000
FUNCTION:DB2UDB, RAS/PD component, pdErrorCatch, probe:30
START   : Error catch set for ZRC -2146435004
```

ZRC -2146435004 是用于锁超时的 DB2 内部错误码。可以通过下面的 db2diag 调用查看这些错误码。

例 9-24 使用 db2diag 查看 DB2 内部错误码的含义：

```
db2diag -rc -2146435004
```

通过使用 db2pdcfg，数据库引擎现在被配置为每当出现锁超时时调用 db2cos 脚本。db2cos 脚本收集判别锁超时原因所需的所有监视器信息。为此，DBA 必须修改 db2cos 脚本，以便用已知的选项调用 db2pd。可以在下面的子目录中找到 db2cos 脚本：

- Windows 环境下，DB2 install directory\BIN\db2cos.bat，例如 C:\Program Files\IBM\SQLLIB\BIN\db2cos.bat；
- UNIX/Linux 环境下，Instance owner home/sqllib/bin/db2cos。

在 Windows 环境下，默认的 db2cos.bat 脚本看上去如下所示：

例 9-25 Windows 上默认 db2cos.bat 的内容：

```
setlocal
:iterargs
if %0. == . goto iterdone
if /i %0. == INSTANCE. set INSTANCE=%1
if /i %0. == DATABASE. set DATABASE=%1
```

```
        if /i %0. == TIMESTAMP. set TIMESTAMP=%1
  if /i %0. == APPID. set APPID=%1
        if /i %0. == PID. set PID=%1
        if /i %0. == TID. set TID=%1
        if /i %0. == DBPART. set DBPART=%1
 if /i %0. == PROBE. set PROBE=%1
        if /i %0. == FUNCTION. set FUNCTION=%1
        if /i %0. == REASON. set REASON=%1
        if /i %0. == DESCRIPTION. set DESCRIPTION=%1
        if /i %0. == DiAGPATH. set DIAGPATH=%1
  shift
 goto iterargs
 :iterdone
 if %DATABASE%. == . goto no database
 db2pd -db %DATABASE% -inst >> %DIAGPATH%\db2cos%PID%%TID%.%DBPART%
 goto exit
 :no_database
  db2pd -inst >> %DIAGPATH%\db2cos%PID%%TID%.%DBPART%
 :exit
```

对于数据库级的事件或故障，默认的 db2cos 脚本用-db 和-inst 选项调用 db2pd。DBA 用一个 db2pd 调用替换相应的行，该调用收集锁超时分析所需的监视器数据。

例 9-26　更改 db2cos 脚本，以收集用于锁超时分析的数据：

```
if %DATABASE%. == . goto no_database
db2pd -db %DATABASE% -locks wait -transactions -agents -applications -dynamic
   >> %DIAGPATH%\db2cos%PID%%TID%.%DBPART%
 goto exit
```

现在，db2cos 脚本已准备好，DBA 可以坐等下一次锁超时事件的发生。

假设像之前描述的那样，用户 A 与 B 之间发生相同的锁情形。但是，这一次设置了 LOCKTIMEOUT，因此过了 10 秒(LOCKTIMEOUT = 10)之后用户 B 的事务被回滚。用户 B 通知 DBA 回滚自己的事务，并且收到 SQL 错误消息-911 和原因码 68(SQL code -911 / reason code 68 = locktimeout)。于是，DBA 检查通过自动调用 db2cos 脚本收集到的监视器数据。

首先，DBA 用锁超时内部错误码调用 db2diag，以确定锁超时发生的确切时间。

例 9-27　在 db2diag.log 中检查锁超时事件的时间点：

```
db2diag -g data:=-2146435004
2007-11-18-14.27.24.656000+060 I6857H409        LEVEL: Event
PID     : 2968              TID  : 2932      PROC : db2syscs.exe
```

```
INSTANCE:DB2                  NODE : 000         DB   : SAMPLE
APPHDL  : 0-21                  APPID: *LOCAL.DB2.061226132544
AUTHID  : FECHNER
FUNCTION:DB2UDB, lock manager, sqlplnfd, probe:999
DATA #1 : <preformatted>
Caught rc -2146435004.  Dumping stack trace.
```

db2diag.log 条目显示，在 2007-11-18-14.27.24.656000 时发生了一次锁超时。由于 db2cos 脚本将它的输出写到%DIAGPATH%中的 db2cos%PID%%TID%.%DBPART% 文件中，DBA 有望在实例的诊断路径中找到一个 db2cos29682932.0 文件：

- %DIAGPATH% = instance's diagnostic path = on Windows by default C:\Program Files\IBM\SQLLIB\DB2
- %PID% = processID= 2968(如 db2diag.log 条目中所示)
- %TID% = threadID= 2932(也显示在 db2diag.log 条目中)
- %DBPART% = database partition = 0(在一个非分区数据库环境中)

文件的内容与本节第一部分中逐步考察的那个 db2pd 监视器输出相似，DBA 可以通过它来识别锁超时的原因。

捕捉到锁超时后，DBA 可以通过-catch clear 选项调用 db2pdcfg 来禁用 db2cos 脚本。

例 9-28　再次使用 db2pdcfg 清除错误捕捉设置：

```
db2pdcfg -catch clear
All error catch flag settings cleared.
```

9.5　事件监视器及监控案例

快照监视器监控的是数据库的实时数据，Event Monitor(事件监视器)记录某事件(event)或转变(transition)出现时某段时间内数据库活动的情况。事件监视器收集监视器数据，例如特定事件或者事务发生。因此，事件监视器提供了一个当事件或者活动发生的时候，不能使用快照监视器监视时收集数据库系统监视器数据的方法。例如，要捕获每当死锁周期发生时的监视器数据。如果您对死锁的概念比较熟悉的话，就应该知道数据库有一个被称之为死锁检测器的特殊进程(db2dlock)，它在后台安静地运行，并且在预定的间隔(dlchktime)时间内会“苏醒”，用于为死锁周期扫描当前正在锁定的系统。如果死锁周期内被发现，死锁检测器将会随机选择、回滚并且终止涉及在此次周期内的任意一个事务。结果，被选择出来的那个事务将会接收到一个 SQL 错误代码(-911)，并且所有实际上已经获得的锁被释放，以便于剩下的事务能够继续执行。像这样的一系列事件的信息将不能被快照监视器所捕获，当死锁出现时，DB2 通过对多个事务中的某个事务发出 ROLLBACK(回退)操作去解

决死锁问题。有关死锁操作的信息也不容易用 Snapshot Monitor 捕获到，因为在能够拍得快照之前，死锁可能已经被解决了。然后，事件监视器却可以捕获该事件的重要信息，因为它可以在死锁周期被检测到的瞬间被激活。这两种监视器的另外一个显著的不同是：快照监视器以后台进程的方式驻留，从一个数据库连接建立就开始捕获监视器数据；相反地，事件监视器必须在它们被使用之前专门去建立和激活。几个不同的事件监视器可以共存，并且每个事件监视器只有在特定类型的事件或者事务发生的时候才会被激活。

Event Monitor 类似其他的数据库对象，因为它们是使用 SQL DDL(数据定义语言)创建的。主要差别是 Event Monitor 可以像 Snapshot Monitor 的开关那样被打开或关闭。

当创建 Event Monitor 时，必须声明要被监视事件的类型。当以下事件发生时，相应事件记录便被记录下来：

DATABASE——当最后一个应用程序与数据库断开时，记录下一个事件记录。

TABLES——当最后一个应用程序与数据库断开时，记录下每个活动表的事件记录。

DEADLOCKS——对于每个死锁，记录下事件记录。

TABLESPACES——当最后一个应用程序与数据库断开时，对每个活动表空间记录下事件记录。

CONNECTIONS——当应用程序与数据库断开时，对每个数据库连接事件记录下事件记录。

STATEMENTS——对于由应用程序发出的每条 SQL 语句(动态或静态)，记录事件记录。

TRANSACTIONS——当事务完成(COMMIT 或 ROLLBACK)时，为该事务记录事件记录。

Event Monitor 的输出存放在目录或命名管道(pipe)中(注：从 DB2 V8 后可以存放在表中，这些事件监控的表可以自动生成)。当 Event Monitor 被激活时，管道和文件的存在将得到证实。如果 Event Monitor 目标位置是一个命名管道，那么提示从管道读出数据是应用程序的职责。如果 Event Monitor 的目标位置是目录，数据流将被写入到一系列文件中。这些文件顺序编号并且都有一个文件附加名 evt(例如 00000000 . evt、00000001 . evt 等)。当定义事件监视器时，规定 Event Monitor Event 文件的大小与数目。

1. 事件监视器的创建方法和步骤

(1) 创建一个 SQL Event Monitor，写入文件：

```
    db2 create event monitor evmname for eventtype write to file 'directory'
    例：db2 create event monitor SQLCOST for deadlocks,statements write to file
'/db2db/event'
```

(2) 激活事件监视器(确保有充足的可用磁盘空间)：

```
$>DB2"set event monitor SQLCOST state = 1"
```

(3) 让应用程序运行。

(4) 取消激活事件监视器：

```
$>DB2"set event monitor SQLCOST state = 0"
```

(5) 使用 DB2 提供的 db2evmon 工具来格式化 SQL Event Monitor 原始数据(根据 SQL 吞吐率可能需要数百兆字节的可用磁盘空间)：

```
$> db2evmon -db DBNAME -evm SQLCOST > sqltrace.txt
```

(6) 浏览整个已格式化的文件，寻找显著大的成本数(一个耗时的过程)：

```
  $> more sqltrace.txt
```

其实在实际的性能调优中，事件监视器并没有被广泛运用，这是因为过去事件监视器的输出只能写到文件或命名管道中，直到 DB2 V8 版本才可以写到表中，使得 DBA 可以利用 SQL 去读取；其次是因为 DB2 的事件监视器在监控期间会产生非常大的文件。

2. 事件监控器案例

下面我们举一个事件监视器存放在表中的例子：

对于数据库管理员，调优数据库常常是一项挑战。调优应用程序是一种方法，但在大多数生产系统中，DBA 很少甚至不能更改源代码，因而也就限制了他们调优应用程序的能力。这在 DBA 使用第三方工具时更是如此。所以，通常最有效的调优方法是解决问题的根源，即从 SQL 语句本身入手。通过查找哪些 SQL 语句消耗的资源最多来获得最佳性能，然后决定采取一定的措施来减少资源消耗。

通常，在第一次安装数据库时，会将其性能调至最优，但随着时间的流逝，一些常用的东西开始变得越来越慢。这在拥有大量数据的系统(例如决策支持系统)中尤为如此。用户开始发现诸如锁升级、全表扫描以及排序这样的因素造成性能下降，这些操作往往会迫使系统访问磁盘而不是访问内存。当出现这种情况时，最好检查一下 SQL，看看可以做哪些改进。

我们举这个事件监控器案例，旨在解决的问题是：针对通常的活动，调优正在用于最频繁访问数据库的 SQL，最消耗资源的 SQL。为了简化如何监控应用程序中 SQL 语句的问题，我们通过 DB2 的事件监视器，确定哪些 SQL 语句消耗的资源最多。

1) 运行事件监视器

首先，必须创建事件监视器，然后运行监视器来收集将要分析的数据。

打开一个新的 DB2 命令行处理器会话，然后执行以下 DB2 命令：

```
db2 => update monitor switches using statement on
db2 => create event monitor sql_trace for statements write to table
```

创建完后，我们可以看到，DB2 自动生成了下面这张表

```
D:\>db2 list tables for all | find /i "stmt"
 STMT_SQL_TRACE                    ORACLE         T
 2008-10-09-15.09.35.781000
 db2 => set event monitor sql_trace state=1
```

(1) 创建事件监视器。

(2) 使该会话一直处于打开状态，直到这些数据库活动完成。请确保 STMT_SQL_TRACE 所在的表空间有足够的存储空间，在此时间有可能会产生大量的数据。表空间的大小取决于用户想要捕获的 SQL 语句的数目和交易量。

(3) 执行正常的数据库活动，直到您想监控的时段结束。这一监控阶段可以是问题产生时期，也可以是通常的数据库高峰期间。

(4) 回到在步骤(1)中所打开的会话，然后发出以下语句：

```
db2 => set event monitor sql_trace state=0
db2 => terminate
```

2) 分析输出

由于已经在数据库表 STMT_SQL_TRACE 中存储了所需要的信息，因此可以查询该表以确定“讨厌”且耗时的 SQL 语句。

须要确定 4 类 SQL 语句。在执行以下查询以确定这些语句之前，在数据库配置参数中，至少要为应用程序堆大小(applheapsz)分配 256 个页面。

- 按照执行时间降序排列执行耗时最长的 SQL 语句。为了确定这些语句，使用下面的 SQL SELECT 语句：

```
select stmt text,(stop time-start time) "ExecutionTime(sec)"
from stmt_sql_trace
where stmt operation not in(7, 8, 9, 19 )
order by decimal(ExecutionTime) desc
fetch first 10 rows only
```

- 按照频率降序排列执行次数最多的 SQL 语句。可以用下面这条查询来确定这些语句：

```
select distinct(stmt_text),count(*) Count from stmt_sql_trace
where operation not in(7,8,9,19)
group by stmt_text
order by count(*)desc
fetch first 10 rows only
```

- 按照 CPU 时间降序排列最耗 CPU 时间的 SQL 语句。用下面这条查询来确定这些语句：

```
select stmt_text ,user_cpu_time "UserCPU(sec)" from stmt_sql_trace
where operation not in(7,8,9,19)
order by usrcpu desc
fetch first 10 rows only
```

- 按照总排序时间降序排列排序时间最长的 SQL 语句。用下面这条查询找到这些语句：

```
select stmt_text ,total_sort_time "TotalSortTime(ms)" from stmt_sql_trace
where operation not in(7,8,9,19 )
order by decimal(total_sort_time) desc
fetch first 10 rows only
```

捕获每一类中的 SQL 语句，并将它们放到 tune.sql 文件中。将下面这行插入到该文件中，这样可以更改工作负载中每条语句的执行频率：

```
--#SET FREQUENCY <x>
```

这里的<x>表示随后要执行 SQL 语句的次数。您的 tune.sql 文件类似于这样：

```
--#SET FREQUENCY 100
SELECT COUNT(*) FROM EMPLOYEE;
SELECT * FROM EMPLOYEE WHERE LASTNAME='HAAS';
--#SET FREQUENCY 1
SELECT AVG(BONUS), AVG(SALARY) FROM EMPLOYEE
GROUP BY WORKDEPT ORDER BY WORKDEPT;
```

将这些 SQL 语句复制到 tune.sql 之后，检查任何 SQL 语句的 WHERE 子句中是否具有参数标志符(?)。将参数标志符改为适当的数据类型值，以便在没有任何错误的情形下执行 SQL 语句。

为了确定哪些索引可能提高性能，按如下脚本执行索引顾问程序：

```
$cd /sapr3/prod
$db2advis -d sample -i tune.sql -t 0 -o tuneidx.sql
```

所有推荐的索引将被放置在文件 tuneidx.sql 中。编辑该文件，在文件开始处添加一条连接语句：

```
connect to sample user userid using password;
```

在该文件末尾添加下面这行：

```
terminate;
```

现在可以运行该文件以创建推荐的索引：

```
$db2 -tf tuneidx.sql -z tuneidx.log
```

注意：

其中，tuneidx.log 捕获 tuneidx.sql 的所有输出。

9.6　db2mtrk 及监控案例

db2mtrk 是用于在 DB2 数据库中进行内存跟踪的工具，可以用于查看实例、数据库、代理进程当前对内存的使用状态。

例 9-29　db2mtrk 监控示例 1。

```
db2mtrk -i -d -v
Memory for database:  SAMPLE
Backup/Restore/Util Heap is of size 16384 bytes
Package Cache is of size 81920 bytes
Catalog Cache Heap is of size 65536 bytes
  Buffer Pool Heap is of size 4341760 bytes
  Buffer Pool Heap is of size 655360 bytes
  Buffer Pool Heap is of size 393216 bytes
  Buffer Pool Heap is of size 262144 bytes
  Buffer Pool Heap is of size 196608 bytes
  Lock Manager Heap is of size 491520 bytes
  Database Heap is of size 3637248 bytes
  Other Memory is of size 16384 bytes
  Application Control Heap is of size 327680 bytes
  Application Group Shared Heap is of size 57344000 bytes
Total: 67829760 bytes
```

例 9-30 db2mtrk 监控示例 2。

```
C:\>db2mtrk -i -d -p -r 1200
在 2008/10/15 10:10:49 跟踪内存
用于实例的内存
 other       fcmbp      monh
 13.3M       768.0K     320.0K
用于数据库 SAMPLE  的内存
 utilh       pckcacheh   other       catcacheh   bph(1)      bph(S32K)
 64.0K       384.0K      128.0K      128.0K      2.3M        832.0K
 bph(S16K)   bph(S8K)    bph(S4K)    shsorth     lockh       dbh
 576.0K      448.0K      384.0K      0           320.0K      12.2M
apph(15)    apph(14)    apph(13)    apph(12)    apph(11)    appshrh
64.0K       64.0K       64.0K       64.0K       64.0K       128.0K
用于代理程序 3276 的内存
other
       192.0K
用于代理程序 2964 的内存
    other
    192.0K
用于代理程序 4128 的内存
    other
     576.0K
用于代理程序 4332 的内存
    other
     192.0K
```

db2mtrk 工具的语法如下：

```
>>-db2mtrk--+----+--+----+--+----+--+---+------------------------>
            '-i-'  '-d-'  '-p-'  +-m-+
                                 '-w-'
>--+------------------------+--+---+--+---+-------------------><
   '-r--interval--+-------+-'  '-v-'  '-h-'
                  '-count-'
```

- db2mtrk -i　　#显示当前实例的内存使用情况
- db2mtrk -i　-v　#显示当前实例的内存使用的详细信息
- db2mtrk -d　　#显示数据库的内存使用情况
- db2mtrk -d -v　#显示数据库的内存使用情况的详细信息
- db2mtrk -p　#显示代理进程专用内存使用率
- db2mtrk -h　#显示帮助信息

-m 参数选项用于显示最大的内存使用上线

-w 参数选项用于显示使用过程中内存达到的最大值，即 watermark

-r 参数选项用于重复显示，其中 interval 是重复显示的时间间隔数，count 是要重复显示的次数

interval 指定以秒为单位的重复的间隔

count 指定间隔的次数

-v 详细输出

-h 显示帮助信息

例 9-30 显示了实例级别和数据库级别内存使用情况的详细信息。

在 DB2 V9.1 中对该命令的-d 选项和-i 选项作了如下更改：

- 在 Windows 平台上，现在支持用于显示数据库级别内存的-d 选项。
- 由于现在可通过-d 选项来显示数据库级别内存，所以-i 选项仅用于显示实例级别内存。

在 db2mtrk 工具的输出的信息中有下面几种类型的信息：

- 当前值的大小
- 最大值的限制(hard limit)
- 最高值(high water mark)
- 类型(用于指定使用了哪种内存)
- 代理进程使用的内容(只针对私有内存池)

注意：

在我们使用 db2mtrk 工具时，主要查看高水位和实际的配置之间是否接近。例如，如果 util_heap_sz 的高水位逼近了实际的数据库配置参数指定的值，那说明您可以考虑增大该参数。

9.7　活动监视器

上面我们介绍的很多工具都是基于命令行的，DB2 UDB V8.2 中增加了一个新的图形化工具，称为活动监视器(Activity Monitor)。它使用各种 SQL 函数和过程获得性能数据，对系统进行分析。从图 9-4 中可以看出，查询执行了两个排序，花费的时间少于 12 秒。Activity Monitor 还提供许多其他报告，可以用来分析 DB2 的性能状况。

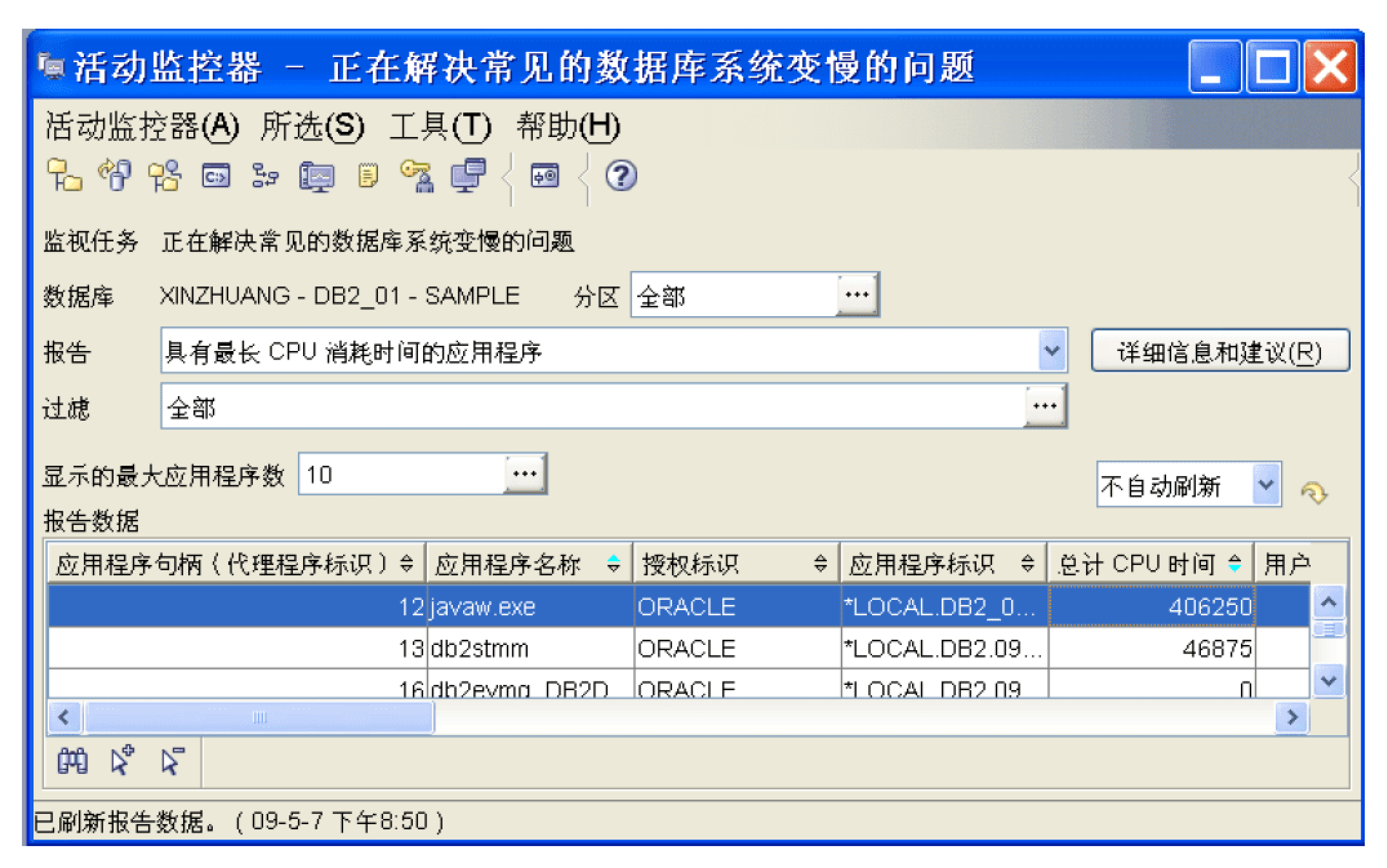


图 9-4 Activity Monitor - 运行时分析

其实，活动监视器本质也是在底层调用一些监控命令和函数，只不过是以图形化的直观方式显现出来。不过，我还是建议大家熟练掌握命令行的方式。一是因为真正的高手是很少用图形化界面的，二是因为在实际的运行环境中往往没有配置图形化环境。不过，活动监视器也不失为一个好的监控维护工具。

9.8 DB2 性能监控总结

以上的各个小节中详细说明了在 DB2 数据库中可以使用的监控工具和手段，针对具体的某些性能指标，可以通过上面提到的一个或多个工具进行监控。

在实际的生产环境下，在怀疑数据库中可能存在着性能问题时，就可以通过上面各节介绍的监控工具和手段来发现数据库中可能存在的瓶颈或不合理的资源使用。在实际发生性能问题时，产生的原因可能并不单一，而往往是由很复杂的原因造成的，所以需要通过多方面的监控分析才能得到一个准确的结论。因而，以上介绍的这些工具经常需要结合起来使用，然后将监控的结果综合地进行分析，才能得到一个最准确的分析结论。

第10章

锁 和 并 发

我们在进行客户支持时遇到最多的话题之一就是锁。“为什么 DB2 锁住了这个表、行或者对象？”,“这个锁会阻塞多长时间及为什么？”,“为什么出现了死锁？”,“我的锁请求在等待什么？”，诸如此类等等。仔细地分析一些常见的锁示例，可以说明 DB2 锁定策略背后的原则。在国内，很多 DB2 用户都会碰到有关锁等待、死锁和锁升级等与锁相关的问题。在本章中，我们将会对这些问题进行详细讲解，并介绍如何解决这些问题。

本章主要讲解以下内容：

- 锁的概念
- 锁的属性、策略和模式
- 隔离级别的概念及使用
- 锁转换、锁等待、锁升级和死锁
- 与锁有关的性能问题
- 锁与应用程序设计
- 锁监控工具及应用案例
- 最大化并发性

10.1 锁的概念

10.1.1 数据一致性

理解数据一致性

什么是数据一致性？我们通过一个示例来回答这个问题。假定您的公司拥有多家连锁饭店，公司用一个数据库来跟踪每家饭店中的货物存储量。为了使货物的采购过程更方便，数据库中包含了每个连锁店的库存表。每当一家饭店收到或用掉一部分货物时，与该饭店

相对应的库存表就会被修改以反映库存变化。

现在，假定从一家店调配若干瓶番茄酱到另一家店。为了准确地表示这一次库存调配，调出方饭店表中存储的番茄酱瓶数必须减少，而接收方饭店表中存储的番茄酱瓶数必须增加。如果用户减少了调出方饭店库存表中的番茄酱瓶数，但没有增加接收方库存表中的番茄酱瓶数，数据就会出现不一致。此时所有连锁店的番茄酱的总瓶数就不准确了。

如果用户忘记了进行所有必要的更改(正如前面示例中的一样)，或者在进行更改的过程中系统崩溃了，又或者数据库应用程序由于某种原因过早地停止了，那么数据库中的数据都就变得不一致。当几个用户同时访问相同的数据库表时，也可能发生不一致。为了防止数据的不一致(尤其是在多用户环境中)，DB2 的设计中集成了下列数据一致性支持机制：

- 事务
- 锁
- 隔离级别

10.1.2 事务和事务边界

事务(也称为*工作单元*)是一种将一个或多个 SQL 操作组合成一个单元的可恢复操作序列，通常位于应用程序进程中。事务的启动和终止定义了数据库的一致性：要么将一个事务中执行的所有 SQL 操作的结果都应用于数据库(提交)，要么完全取消并丢弃已执行的所有 SQL 操作的结果(回滚)。

运行嵌入式 SQL 应用程序或脚本，在可执行 SQL 语句第一次执行时(在建立与数据库的连接之后或在现有事务终止之后)，事务就会自动启动。在启动事务之后，必须由启动事务的用户或应用程序显式地终止它，除非使用了称为自动提交(automatic commit)的过程(在这种情况下，发出的每个单独的 SQL 语句被看做单个事务，它一执行就被隐式地提交了)。

在大多数情况下，通过执行 COMMIT 或 ROLLBACK 语句来终止事务。当执行 COMMIT 语句时，自从事务启动以来对数据库所做的一切更改就成为永久性的了——即它们被写到磁盘。当执行 ROLLBACK 语句时，自从事务启动以来对数据库所做的一切更改都被撤销，并且数据库返回到事务开始之前所处的状态。不管是哪种情况，数据库在事务完成时都保证能回到一致状态。

一定要注意一点：虽然事务通过确保对数据的更改仅在事务被成功提交之后才成为永久性的，从而提供了一般的数据库一致性，但还是须要用户或应用程序来确保每个事务中执行的 SQL 操作序列始终会导致一致的数据库。

1. COMMIT 和 ROLLBACK 操作的效果

正如前面提到的，通常通过执行 COMMIT 或 ROLLBACKSQL 语句来终止事务。为了理解这些语句如何工作，我们看下面的示例。

如果按所示的顺序执行下列 SQL 语句(由 3 个事务组成的简单工作负载)：

```
CONNECT TO SAMPLE
CREATE TABLE DEPARTMENT (DEPT_ID INTEGER NOT NULL, DEPT_NAME VARCHAR(20))
INSERT INTO DEPARTMENT VALUES(100, 'PAYROLL')
INSERT INTO DEPARTMENT VALUES(200, 'ACCOUNTING')
COMMIT

INSERT INTO DEPARTMENT VALUES(300, 'SALES')
ROLLBACK

INSERT INTO DEPARTMENT VALUES(500, 'MARKETING')
COMMIT
```

这将创建一个名为 DEPARTMENT 的表，它的结构如表 10-1 所示。

表 10-1　表 DEPARTMENT 的结构

DEPT_ID	DEPT_NAME
100	PAYROLL
200	ACCOUNTING
500	MARKETING

当执行第一个 COMMIT 语句时，创建名为 DEPARTMENT 的表并向表中插入两条记录，这两个操作都会变成永久性的。当执行到 ROLLBACK 语句时，删除插入 DEPARTMENT 表中的第三条记录，该表返回到执行插入操作之前所处的状态。最后，当执行到第二个 COMMIT 语句时，插入到 DEPARTMENT 中的第四条记录成为永久性的，而数据库再次返回到一致状态。

从上面这个示例中可以看出，提交或回滚操作只影响在这个操作所结束的事务内作出的更改。只要数据更改仍然未被提交，其他用户和应用程序通常就无法看见它们(也有例外情况，本章的最后我们将进行讨论)，并可以通过执行回滚操作取消它们。但是，一旦数据更改被提交了，其他用户和应用程序就可以访问它们，并且再也不能通过回滚操作取消它们了。

2. 不成功事务的效果

我们刚才已看到当通过 COMMIT 或 ROLLBACK 语句终止事务时会发生什么。但是，如果在事务完成前出现系统故障，那会发生什么情况呢？在这种情况下，DB2 数据库管理程序会取消所有未提交的更改，从而恢复数据库一致性(假定在事务启动时就存在这样的一致性)。图 10-1 对比了成功的事务和在成功终止之前失败的事务的效果。

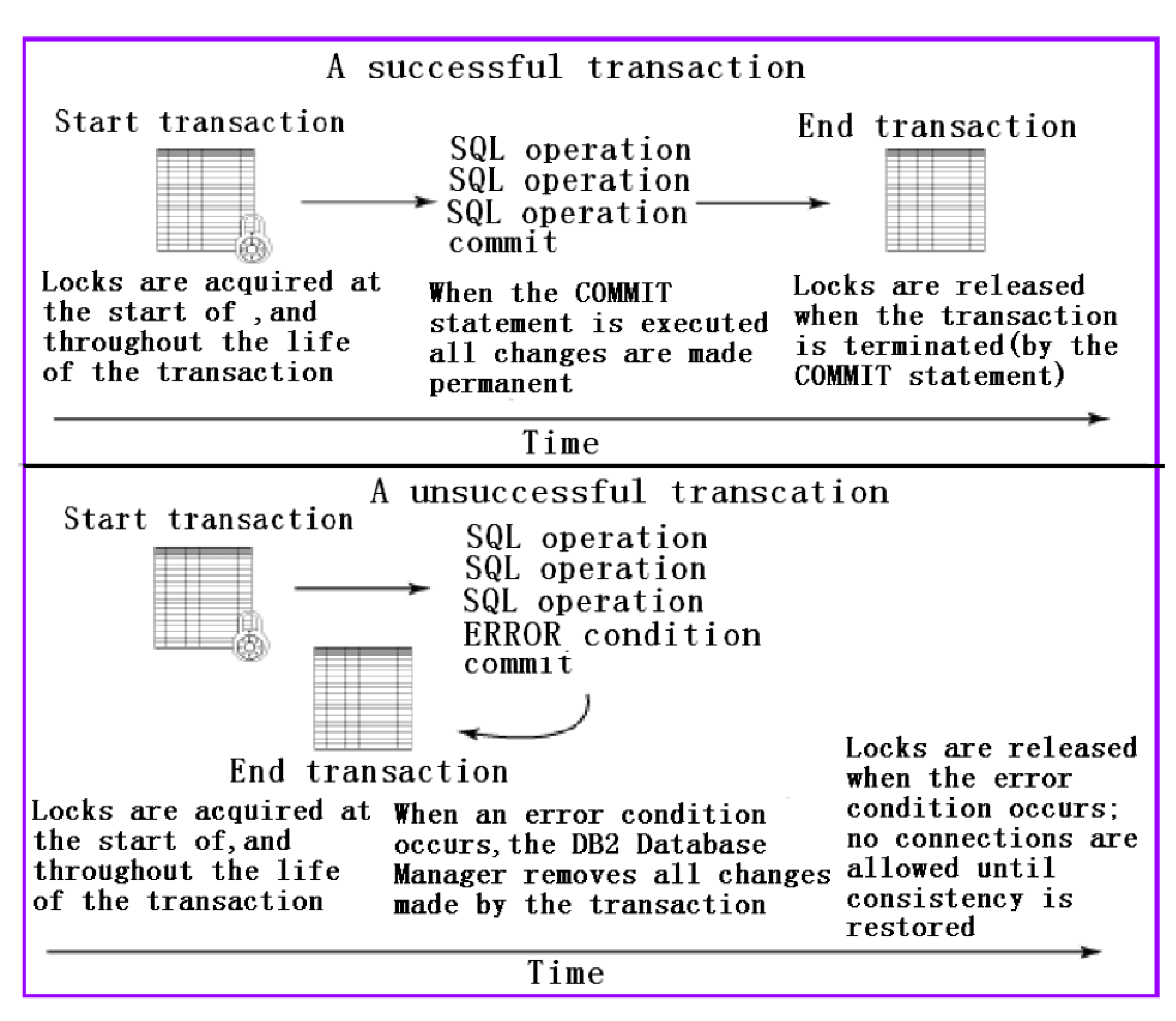


图 10-1　对比成功的和不成功的事务

10.1.3　锁的概念

在关系数据库(DB2、Oracle、Sybase、Informix 和 SQL Server 等)中，最小的恢复和交易单位为事务(Transactions)，事务具有 ACID(原子性、一致性、隔离性和永久性)特征。关系数据库为了确保并发用户在存取同一数据库对象时的正确性(即无丢失更新、可重复读、不读"脏"数据，无"幻像"读)，数据库中引入了并发(锁)机制。

一个成熟的关系数据库系统，应该能够允许多个应用程序同时对相同数据进行访问。当这种情况发生时，要保证数据库的完整性，就必须要有一定的机制用于控制数据记录的

读取、插入、删除和更新。锁就是这样一种机制。基本的锁有两种类型：排它锁(Exclusive locks，记为 X 锁)和共享锁(Share locks，记为 S 锁)。

锁是一种用来将数据资源与单个事务关联起来的机制，其用途是当某个资源与拥有它的事务关联在一起时，控制其他事务如何与该资源进行交互。我们称与被锁定资源关联的事务持有或拥有该锁，DB2 数据库管理程序用锁来禁止事务访问其他事务写入的未提交数据(除非使用了未提交的读隔离级别)，并禁止其他事务在拥有锁的事务使用限制性隔离级别时对这些行进行更新。一旦获取了锁，在事务终止之前，就一直持有该锁；该事务终止时释放锁，其他事务就可以使用被解锁的数据资源了。

如果一个事务尝试访问数据资源的方式与另一个事务所持有的锁不兼容(稍后我们将研究锁兼容性)，则该事务必须等待，直到拥有锁的事务终止为止。这被称为锁等待。当锁等待事件发生时，尝试访问数据资源的事务所做的只是停止执行，直到拥有锁的事务终止或不兼容的锁被释放为止。

举个例子，假如事务 T 对数据 D 加 X 锁，则其他任何事务都不能再对 D 加任何类型的锁，直至 T 释放 D 上的 X 锁；一般要求在修改数据前要向该数据加排它锁，所以排它锁又称为写锁。若事务 T 对数据 D 加 S 锁，则其他事务只能对 D 加 S 锁，而不能加 X 锁，直至 T 释放 D 上的 S 锁；一般要求在读取数据前要向该数据加共享锁，所以共享锁又称为读锁。我们可以通过调整数据库的加锁策略，来适应一定的并发性需求。

通过对数据库对象加锁，我们可以避免以下情况的发生：

- 由于并发更改造成数据的丢失

如果应用程序在一个事务中多次执行同一 SQL 语句，而且后续执行会返回附加行，就是发生了“幻像读”。

图 10-2 中说明了这种场景。假设航空机票代理 1 查询 512 航班的所有空闲座位，发现只有一个座位空着。在此之后，由于一位乘客取消了飞机票，航空机票代理 2 取消了这个航班上一个座位的预订。如果代理 1 在这个事务中再次执行同一查询，那么它不会得到与上次查询完全相同的数据集——因为出现了新的座位。

让我们用数据库来对上述情况再进行一下细致描述：

P Read 和 Instuct 两人同时来到不同的航空代理处购买 512 航班的机票，两个代理都想为自己的顾客预订 7A 座位，然后两个代理同时输入下列命令：

代理 1：

```
    update reservations set p_name='P Read' where flight='512' and seat='7A'
and  p_name is null;:
```

Reservations

Flight	Seat	P_Name
512	7B	——
512	7A	P Read
⋮	⋮	

1 Select seat
from Reservations
where P_name
where is NULL

2 Update Reservations
Set P_name = "NULL"
Where flight = 512
and Seat = "7A"
and P_Name = "P Read"

512	7B	——

3 Repeat 1
now 7A is
now available

图 10-2 幻像读

代理 2：

```
    update reservations set p_name='Instuct' where flight='512' and seat='7A'
and  p_name is null;:
```

如果没有一定的机制来阻止对同一数据的并发更改，则两个代理都会收到更改成功的信息。P Read 和 Instuct 将会在机场出现并认为他们都已经预订好座位。

如果我们使用了加锁机制进行控制，这种情况可以避免。当 P READ 的代理在更新座位信息的时候，在表中先使用排它锁对数据进行锁定，然后再更新，更新结束再将锁释放，那么 Instuct 的代理就不可能获得同样的座位信息了。

- 读取了未提交的数据(脏读)

当可以读取未提交的数据时可能会出现“脏读”，因为如果这些数据的修改被回滚，就会导致在数据处理中使用无效数据。

图 10-3 说明了这种场景。在这里，更新了一行并更改了 P-Name 字段，但是没有提交。然后，另一个使用 UR 隔离级别的应用程序执行 SELECT 语句。因为 UR 忽略行上的锁，所以它将返回未提交的值。但是，更新事务在提交之前被回滚。因而，SELECT 语句返回错误的值。

图 10-3　读取了未提交的数据

未提交数据是指被命令所更改，但还未被提交的数据。在很多情况下，如果允许用户对未提交的数据进行存取而该数据最终未被提交，用户依据其作出的决定有可能是错误的。

想象如下场景：

在 512 航班上 7C 是唯一剩下的座位，现在 Instuct 来到某个航空代理处想购买 512 航班的机票，该代理发现 7C 是唯一剩下的座位。于是键入下列语句：

```
    update reservations set p_name='Instruct' where flight=512 and seat='7C'
and p_name=NULL;:
```

这时，P Read 来到另外一个航空代理处想购买 512 航班的机票，该代理输入下列查询命令：

```
    select seat from reservations where p_name is NULL;:
```

如果不阻止对未提交数据的读取，则由于 7C 已经被 Instruct 预订了，则该语句返回的记录数为零，意味着该航班已经客满，P READ 就不能买到座位。

过了一会儿，Instuct 认为该航班太贵，于是，该事务相应地回滚，7C 座位被空出。如果数据库通过加锁机制阻止对未提交数据的存取，P Read 就可以买到 7C 座位。

DB2 可以通过加锁机制来阻止对未提交数据的访问。当然，在某些情况下，对未提交数据的访问应该是允许的。

- 防止不可重复读

如果相同查询在同一工作单元中返回不同的结果集，就是出现了“不可重复读”。图 10-4 中说明了这种场景。假设航空机票代理 1 查找从 Dallas 到 Honolulu 的所有航班，航空机票代理 2 在 flight 表中删除了一个航班，因为这个航班被取消了。如果航空机票代理 1 在它的事务中再次执行同样的查询，就不会得到完全相同的数据集：其中不包含取消的那个航班。因为在航空机票代理 1 运行其事务时，允许航空机票代理 2 更改这个表，这个查询将是不可重复读的。

FLIGHT	SEAT	NAME	DESTINATION	ORIGIN
512	7C		DENVER	DALLAS
⋮		—		
⋮				
814	8A	—	SAN JOSE	DENVER
⋮				
134	1C	—	HONOLULU	SAN JOSE
⋮				⋮

Book a flight from Dallas to Honolulu

图 10-4 不可重复读

不可重复读是指由于在同一事务中，两次执行同样的 Select 语句，得到的结果不同。这种情况会导致原先作出的决定由于条件的更改而产生偏差。

例如表 10-2 所示的航班表：

表 10-2 航 班 表

航 班	座 位	旅客名称	目的地	出发地
512	7C		DENVER	DALLAS
814	8A		SAN JOSE	DENVER
134	1C		HONOLULU	SAN JOSE

想象以下场景：

Instuct 需要从 DALLAS 飞往 HONOLULU，他找到航空代理处 1，该代理发现没有直达航班，但可以从 DALLAS 到 DENVER，再到 SAN JOSE，最后到达 HONOLULU 的路

线(如表 10-2)。Instuet 正在决定该路线是否可行时，P Read 从另外一个航空代理处 2 预订了 814 航班的 8A 座位，而这恰好是该航班的最后一个座位。如果这时 Instruct 发现代理提供的路线可行，决定购买时，却发现该路线已经不能成立，Instruct 就必须重新选择路线。

采用加锁机制，DB2 能够支持可重复读请求，它可以允许或阻止应用程序修改其他应用程序正在访问的数据。对于上述场景而言，就是当 Instuct 在做决定时，禁止航空代理处 2 访问和更改 814 航班的 8A 座位信息，因为代理处 1 已经加了锁。

10.2　锁的属性、策略及模式

10.2.1　锁的属性

所有的锁都具有下列基本属性：

- **object**：该属性标识要锁定的数据资源。DB2 数据库管理程序在需要时锁定数据资源(如表空间、表和行)。DB2 支持对表空间、表、行和索引加锁(大型机上的 DB2 还支持对数据页加锁)来保证数据库的并发完整性。不过，在考虑用户应用程序的并发性的问题上，通常并不检查用于表空间和索引的锁。分析该类问题的焦点在于表锁和行锁。
- **size**：该属性指定要锁定的数据资源部分的物理大小。锁并不总是必须控制整个数据资源。例如，DB2 数据库管理程序可以让应用程序独占地控制表中的特定行，而不是让该应用程序独占地控制整个表。
- **duration**：该属性指定持有锁的时间长度。事务的隔离级别通常控制着锁的持续时间。
- **mode**：该属性指定允许锁的拥有者执行的访问类型，以及允许并发用户对被锁定数据资源执行的访问类型。这个属性通常称为*锁状态*。

10.2.2　加锁策略

DB2 可以只对表进行加锁，也可以对表和表中的行进行加锁。如果只对表进行加锁，则表中所有的行都受到同等程度的影响。如果加锁的范围针对表及下属的行，则在对表加锁后，相应的数据行上还要加锁。应用程序究竟是对表加行锁还是同时加表锁和行锁，是由应用程序执行的命令和系统的隔离级别确定的。如果某个应用程序挂起某个数据库对象上的锁定，那么另一个应用程序就可能不能访问该对象。因此，锁定最少量数据并使这些数据不可访问的行级别锁定相比表级别而言，对于最大化并行性更好。但是，锁定需要内存和处理时间，因此单个表锁定可以最小化锁定开销。

10.2.3 锁的模式

DB2 在表一级加锁可以使用表 10-3 所示的方式。

表 10-3 表一级的加锁方式

名称缩写	全 名	描 述
IN	无意图(Intenet None)锁 不需要行锁	该锁的拥有者可以读表中的任何数据，包括其他事务尚未提交的数据，但不能对表中的数据进行更改
IS	意图共享(Intent Share)锁 需要行锁配合	该锁的拥有者在拥有相应行上的 S 锁时可以读取该行的数据，但不能对表中的数据进行更改
IX	意图排它(Intent eXclusive)锁需要行锁配合	该锁的拥有者在拥有相应行的 X 锁时可以更改该行的数据
SIX	共享携意图排它 (Share with Intent eXclusive)锁 需要行锁配合	该锁的拥有者可以读表中的任何数据，如果在相应的行上能够获得 X 锁，则可以修改该行。SIX 锁的获得比较特殊，它是在应用程序已经拥有 IX 锁的情况下请求 S 锁，或者是在应用程序已经拥有 S 锁的情况下请求 IX 锁时生成的
S	共享(Share)锁 不需要行锁配合	该锁的拥有者可以读表中的任何数据，如果表被加上 S 锁，该表中的数据就只能被读取，不能被改变
U	更新(Update)锁 不需要行锁配合	该锁的拥有者可以读表中的任何数据，在升级到 X 锁之后，还可以更改表中的任何数据。该锁是处于等待对数据进行更改的一种中间状态
X	排它(eXclusive)锁 不需要行锁配合	该锁的拥有者可以读取或更改表中的任何数据。如果对表加上 X 锁，除了未提交读程序外，其他应用程序都不能对该表进行存取
Z	超级排他 (Super Exclusive)锁 不需要行锁配合	该锁不是通过应用程序中的 DML 语言来生成的。一般是通过对表进行删除(Drop)和转换(Alter)操作或创建和删除索引而获得的。如果对表加上 Z 锁，其他应用程序(包括未提交读程序)都不能对该表进行存取

下面对表 10-3 中几种表锁的模式作进一步阐述：

IS、IX、SIX 方式用于表一级并需要行锁配合，它们可以阻止其他应用程序对该表加上排它锁。

- 如果一个应用程序获得某表的 IS 锁，该应用程序可获得某一行上的 S 锁　用于只读操作，同时其他应用程序也可以读取该行，或是对表中的其他行进行更改。
- 如果一个应用程序获得某表的 IX 锁，该应用程序可获得某一行上的 X 锁　用于更改操作，同时其他应用程序可以读取或更改表中的其他行。
- 如果一个应用程序获得某表的 SIX 锁，该应用程序可以获得某一行上的 X 锁　用于更改操作，同时其他应用程序只能对表中的其他行进行只读操作。

S、U、X 和 Z 方式用于表一级，但并不需要行锁配合，是比较严格的表加锁策略。

- 如果一个应用程序得到某表的 S 锁，该应用程序可以读表中的任何数据，同时它允许其他应用程序获得该表上的只读请求锁。如果有应用程序需要更改或读该表上的数据，就必须等 S 锁被释放。
- 如果一个应用程序得到某表的 U 锁，该应用程序可以读表中的任何数据，并最终可以通过获得表上的 X 锁来得到对表中任何数据的修改权。其他应用程序则只能读取该表中的数据。U 锁与 S 锁的区别主要在于更改意图上。U 锁的设计主要是为了避免两个应用程序在拥有 S 锁的情况下同时申请 X 锁而造成死锁。
- 如果一个应用程序得到某表上的 X 锁，该应用程序可以读或修改表中的任何数据。其他应用程序不能对该表进行读或更改操作。
- 如果一个应用程序得到某表上的 Z 锁，该应用程序可以读或修改表中的任何数据。其他应用程序，包括未提交读程序都不能对该表进行读或更改操作。

注意：

对于 IN、IX、IS 和 SIX 这些意图(intent)锁，读者可以这样理解，严格来说它们并不是一种锁，而是存放表中行锁的信息。举个通俗的例子，我们去住一个酒店。可以把整个酒店比喻成一张表，每个房间是一行。那么当我们预定一个房间时，就对该行(房间)加 X 锁，但是同时会在酒店的前台对该行(房间)做一个信息登记(旅客姓名、住多长时间等)。大家可以把意图锁当成是这个酒店前台，它并不是真正意义上的锁，它维护表中每行的加锁信息，是共用的。后续的旅客通过酒店前台来看哪个房间是可住的，那么，如果没有意图锁，会出现什么情况呢？假设我要预订房间，那么每次我都须要到每一个房间查看以确认这个房间有没有住旅客，显然这样做的效率是很低下的。其实，最早的 DB2 版本是没有意图锁的，但是这对并发影响非常大，后来就增加了意图锁。所有的数据库(Oracle、Infomix 和 Sybase)都有意图锁的实现机制。

IN 锁用于表上以允许未提交读。

除了表锁之外，DB2 还支持表 10-4 所示的几种方式的行锁。

表 10-4　DB2 支持的行锁

名称缩写	全　　名	需要表锁的最低级别	描　　述
S	共享(Share)锁	IS	该行正被某个应用程序读取，其他应用程序只能对该行进行读操作
U	更改(Update)锁	IX	某个应用程序正在读该行并有可能更改该行，其他应用程序只能读该行
X	排它(eXclusive)锁	IX	该行正被某个应用程序更改，其他应用程序不能访问该行
W	弱排它(Weak eXclusive)锁	IX	当一行数据被插入表中的时候，该行会被加上 W 锁。锁的拥有者能够更改该行，该锁与 X 锁基本相同，但它与 NW 锁兼容
NS	下一键共享(Next Key Share)锁	IS	锁的拥有者和其他程序都可以读该行，但不能对该行进行更改。当应用程序处于 RS 或 CS 隔离级别下，该锁可用来替代 S 锁
NX	下一键排它(Next Key eXclusive)锁	IX	当一行数据被插入到索引中或从索引中被删除的时候，该行的下一行上会被加上该锁。锁的拥有者可以读，但不能更改锁定行。该锁与 X 锁类似，但它与 NS 锁兼容
NW	下一键弱排它(Next Key Weak eXclusive)锁	IX	当一行被插入到索引中的时候，该行的下一行会被加上该锁。锁的拥有者可以读但不能更改锁定行。该锁与 X 和 NX 锁类似，但它与 W 和 NS 锁兼容

10.2.4　如何获取锁

大多数情况下，DB2 数据库管理程序在需要锁时会隐式地获取它们，因此这些锁在 DB2 数据库管理程序的控制之下。除了使用未提交读隔离级别的情况外，事务从不需要显式地请求锁。事实上，唯一有可能被事务显式锁定的数据库对象是表(lock table)。图 10-5 说明了如何确定所引用的对象获取的锁的类型。

DB2 数据库管理程序总是尝试获取行级锁。但是，可以通过执行特殊形式的 ALTER TABLE 语句来修改这种行为，如下所示：

```
ALTER TABLE [TableName] LOCKSIZE TABLE
```

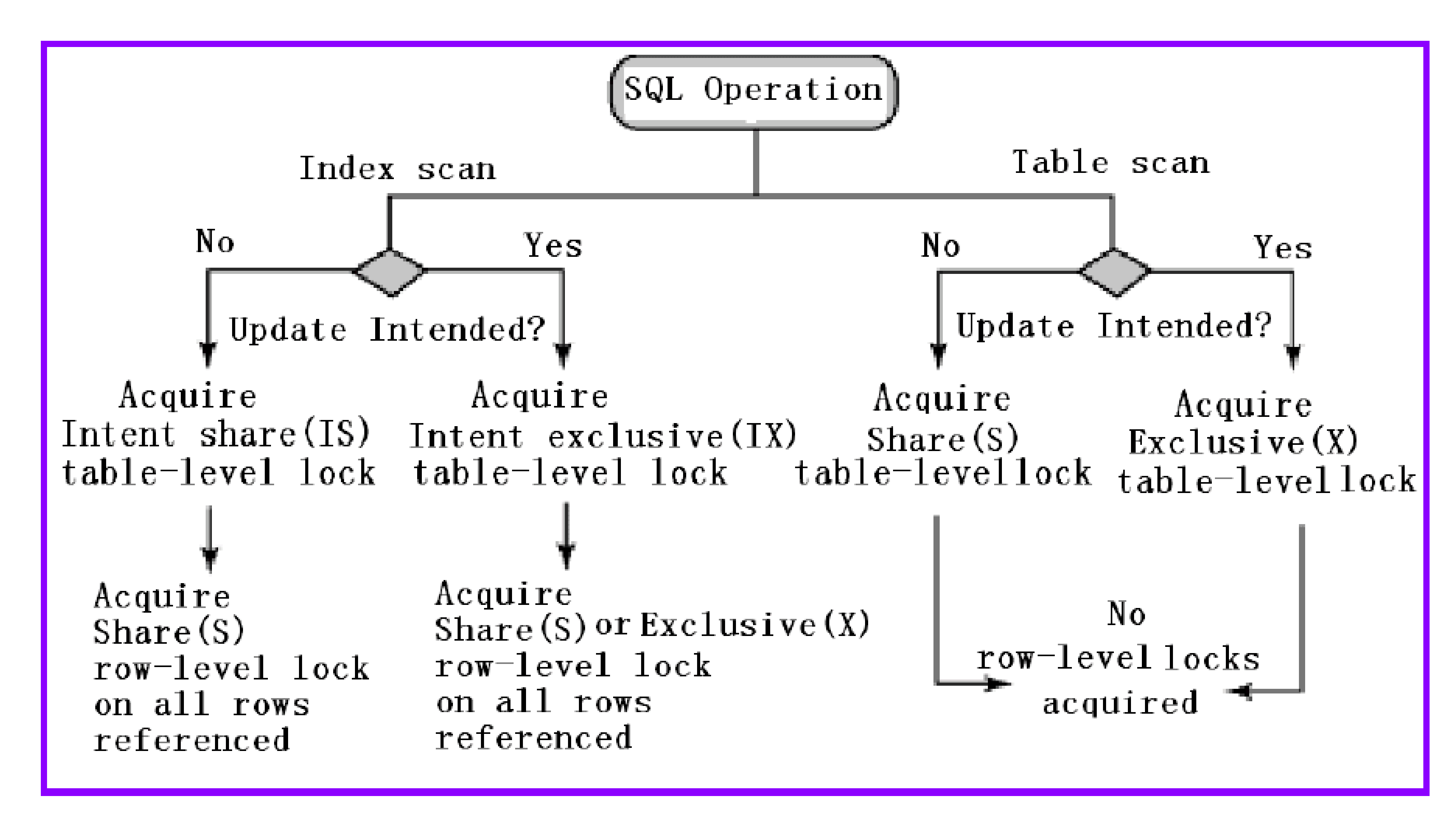


图 10-5　如何确定获取锁的类型

其中的 *TableName* 标识一个现有表的名称，所有事务在访问它时都要获取表级锁。

也可以通过执行 LOCK TABLE 语句，强制 DB2 数据库管理程序为特定事务在表上获取表级锁，如下所示：

```
LOCK TABLE [TableName] IN [SHARE | EXCLUSIVE] MODE
```

其中的 *TableName* 标识一个现有表的名称，对于这个表应该获取表级锁(假定其他事务在该表上没有不兼容的锁)。如果在执行这个语句时指定了共享(SHARE)模式，就会获得一个允许其他事务读取(但不能更改)存储在表中的数据的表级锁；如果执行时指定了互斥(EXCLUSIVE)模式，就会获得一个不允许其他事务读取或修改存储在表中的数据的表级锁。

ALTER TABLE 语句的 LOCKSIZE 子句指定行级别、或表级别的锁定的作用域(详细程度)。默认情况下使用的是行锁定。由这些已定义的表锁定仅请求 S(共享)和 X(互斥)锁定。ALTER TABLE 语句的 LOCKSIZE ROW 子句不会阻止正常的锁定升级。

在下列情况下，由 ALTER TABLE 语句定义的永久表锁定可能比使用 LOCK TABLE 语句获得的单个事务表锁定更可取：

- 表是只读的，并且始终只需要 S 锁定。其他用户也可以获取表的 S 锁定：
- 表通常由只读应用程序访问，但有时也需要单个用户访问以进行简单维护，该用户需要 X 锁定。当维护程序运行时，将只读应用程序锁定在外，但在其他情况下，只读应用程序可以使用最小锁定开销的同时访问表。

ALTER TABLE 语句在全局指定锁定，它影响访问该表的所有应用程序和用户。单个应用程序可以使用 LOCK TABLE 语句来指定应用程序级别的表锁定。

10.2.5 锁的兼容性

在一个应用程序已经对某个对象锁定的情况下，另一个应用程序请求对同一个对象的锁定，此时就会出现锁定兼容性问题。当两种锁定方式兼容时，可以同意对该对象的第二个锁定请求。如果请求的锁定的锁定方式与已挂起的锁定方式不兼容，那么就不能同意第二个锁定请求。相反，请求要等到第一个应用程序释放其锁定，并且释放所有其他现有的不兼容锁定为止。

表 10-5 和表 10-6 显示了锁对象兼容的有关信息。在某些状态下，当另一个进程挂起或正在请求对同一个资源的锁定时，可以同意锁定请求。否则请求者就必须等待，直到所有不兼容的锁定被其他进程释放为止。

注意，当请求者正等待锁定时，可能出现超时。同意该锁定，除非更早的请求者正在等待该资源。

表 10-5 锁的兼容性

锁 A 的方式	锁 B 的方式							
	IN	IS	S	IX	SIX	U	X	Z
IN	兼容	兼容	兼容	兼容	兼容	兼容	兼容	不兼容
IS	兼容	兼容	兼容	兼容	兼容	兼容	不兼容	不兼容
S	兼容	兼容	兼容	不兼容	不兼容	兼容	不兼容	不兼容
IX	兼容	兼容	不兼容	兼容	不兼容	不兼容	不兼容	不兼容
SIX	兼容	兼容	不兼容	不兼容	不兼容	不兼容	不兼容	不兼容
U	兼容	兼容	兼容	不兼容	不兼容	不兼容	不兼容	不兼容
X	兼容	不兼容	不兼容	不兼容	不兼容	不兼容	不兼容	不兼容
Z	不兼容	不兼容	不兼容	不兼容	不兼容	不兼容	不兼容	不兼容

表 10-6 行锁的兼容性

锁 A 的模式	锁 B 的模式						
	S	U	X	W	NS	NX	NW
S	兼容	兼容	不兼容	不兼容	兼容	不兼容	不兼容
U	兼容	不兼容	不兼容	不兼容	兼容	不兼容	不兼容
X	不兼容	不兼容	不兼容	不兼容	不兼容	不兼容	不兼容
W	不兼容	不兼容	不兼容	不兼容	不兼容	不兼容	兼容
NS	兼容	兼容	不兼容	不兼容	兼容	兼容	兼容
NX	不兼容	不兼容	不兼容	不兼容	兼容	不兼容	不兼容
NW	不兼容	不兼容	不兼容	兼容	兼容	不兼容	不兼容

10.3　隔离级别(Isolation Levels)

维护数据库的一致性和数据完整性，同时又允许多个应用程序同时访问同一数据，这样的特性称为并发性。DB2 数据库用来尝试强制实施并发性的方法之一是通过使用隔离级别，它决定在第一个事务访问数据时，如何对其他事务锁定或隔离该事务所使用的数据。DB2 使用下列隔离级别来强制实施并发性：

- 可重复读(Repeatable Read)
- 读稳定性(Read Stability)
- 游标稳定性(Cursor Stability)
- 未提交读(Uncommitted Read)

可重复读隔离级别可以防止所有现象，但是会大大降低并发性(可以同时访问同一资源的事务数量)。未提交读隔离级别提供了最大的并发性，但是“脏读”、“幻像读”和“不可重复读”都可能出现。默认的隔离级别是 CS。

10.3.1　可重复读(RR—Repeatable Read)

可重复读隔离级别是最严格的隔离级别。在该隔离级别下，一个事务的影响完全与其他并发事务隔离，脏读、不可重复的读、幻像读现象都不会发生。当使用可重复读隔离级别时，在事务执行期间会锁定该事务以任何方式引用的所有行。因此，如果在同一个事务中发出同一个 SELECT 语句两次或更多次，那么产生的结果数据集总是相同的。因此，使用可重复读隔离级别的事务可以多次检索同一行集，并对它们执行任意操作，直到提交或回滚操作终止该事务。但是，在事务存在期间，不允许其他事务执行会影响这个事务正在访问的任何行的插入、更新或删除操作。为了确保这种行为不会发生，锁定该事务所引用的每一行——而不是仅锁定被实际检索或修改的那些行。因此，如果一个事务扫描了 1000 行，但只检索 10 行，那么它所扫描的 1000 行(而不仅是被检索的 10 行)都会被锁定。

那么在现实环境中可重复读隔离级别是如何工作的呢？假定如家酒店使用 DB2 数据库跟踪如家酒店的客房信息，包括房间预订和房价信息，还有一个基于 Web 的应用程序，它允许顾客按“先到先服务”的原则预订房间。如果旅馆预订应用程序是在可重复读隔离级别下运行的，当顾客扫描某个日期段内的可用房间列表时，您(旅馆经理)将无法更改那些房间在指定日期范围内的房价。同样，其他顾客也无法进行或取消将会更改该列表的预订(直到第一个顾客的事务终止为止)。图 10-6 说明了这种行为。

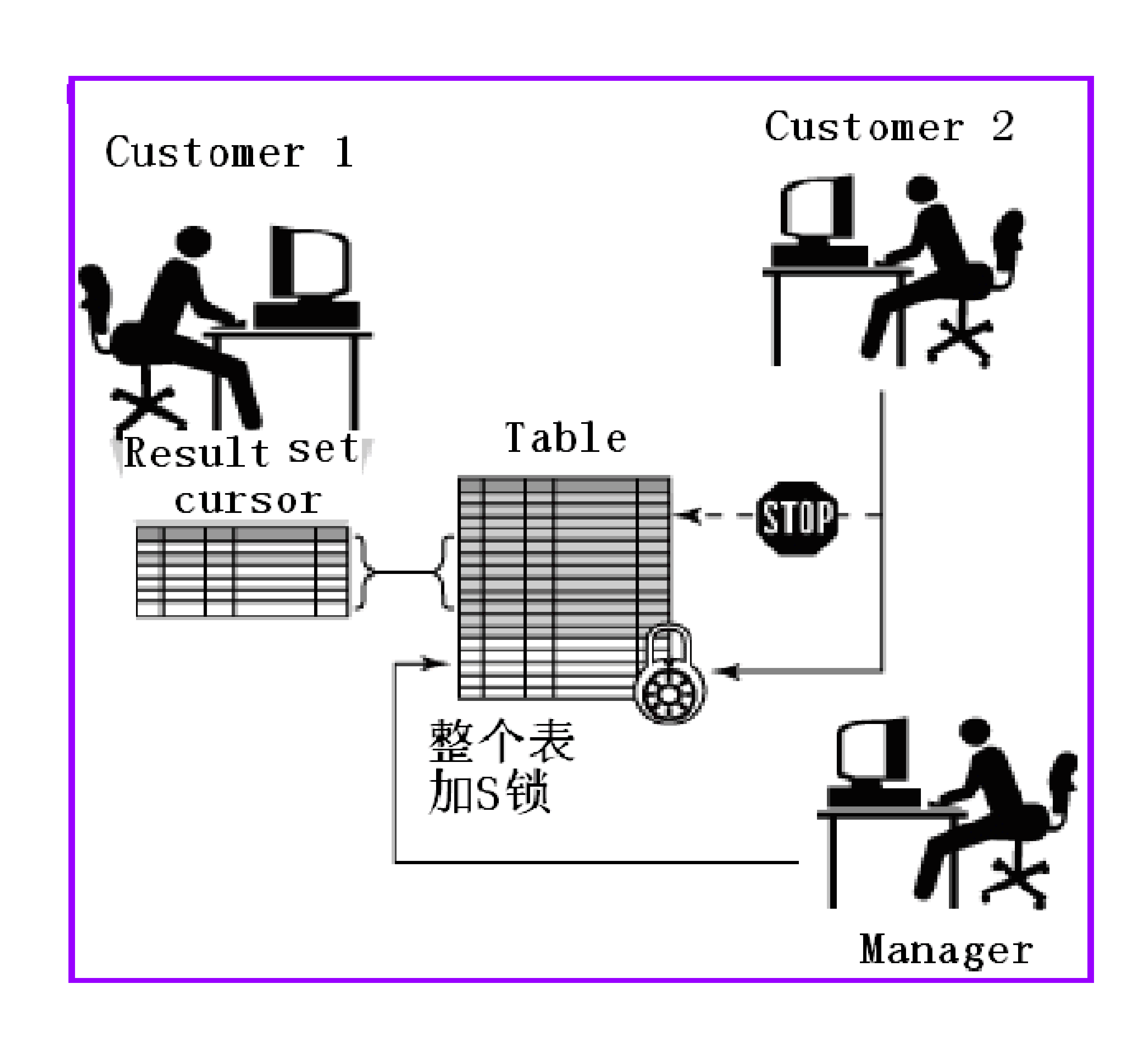


图 10-6　可重复读隔离级别的示例

在使用可重复读隔离级别时，一个事务中所有被读取过的行上都会被加上 S 锁，直到该事务被提交或回滚，行上的锁才会被释放。这样可以保证在一个事务中即使多次读取同一行，得到的值也不会改变。另外，在同一事务中即使以同样的搜索标准重新打开已被处理过的游标，得到的结果集也不会改变。可重复读相对于读稳定性而言，加锁的范围更大。对于读稳定性，应用程序只对符合要求的所有行加锁；而对于可重复读，应用程序将对所有被扫描过的行都加锁。

可重复读(RR)会锁定应用程序在工作单元中引用的所有行。利用“可重复读”，在打开游标的相同工作单元内一个应用程序发出一个 SELECT 语句两次，每次都返回相同的结果。利用“可重复读”，不可能出现丢失更新、脏读和幻像读的情况。

在该工作单元完成之前，“可重复读”应用程序可以尽可能多次地检索和操作这些行。但是，在该工作单元完成之前其他应用程序均不能更新、删除或插入可能会影响结果表的行。“可重复读”应用程序不能查看其他应用程序的未落实更改。

10.3.2　读稳定性(RS—Read Stability)

读稳定性隔离级别没有可重复读隔离级别那么严格；因此，它没有将事务与其他并发事务的效果完全隔离。读稳定性隔离级别可以防止脏读和不可重复读，但是可能出现幻像读。在使用这个隔离级别时，只是锁定事务实际检索和修改的行。因此，如果一个事务扫描了 1000 行，但只检索 10 行，则只有被检索的 10 行(而不是所扫描的 1000 行)被锁定。因此，如果在同一个事务中发出同一个 SELECT 语句两次或更多次，那么每次产生的结果

数据集可能不同。

与可重复读隔离级别一样，在读稳定性隔离级别下运行的事务可以检索一个行集，并可以对它们执行任意操作，直到事务终止。在这个事务存在期间，其他事务不能执行那些会影响这个事务检索到的行集的更新或删除操作；但是其他事务可以执行插入操作。如果插入的行与第一个事务的查询的选择条件匹配，那么这些行可能作为幻像出现在后续产生的结果数据集中。其他事务对其他行所作的更改，在提交之前是不可见的。

那么，读稳定性隔离级别会如何影响如家酒店客房预定应用程序的工作方式呢？当一个顾客检索某个日期段内的所有可用房间列表时，您可以更改这个顾客的列表之外的任何房间的房价。同样，其他顾客可以进行或取消房间预订。如果第一个顾客再次运行同样的查询，其他顾客的操作可能会影响他获得的可用房间列表。如果第一个顾客再次查询同一个日期段内的所有可用房间列表，产生的列表中有可能包含新的房价或第一次产生列表时不可用的房间。图 10-7 说明了这种行为。

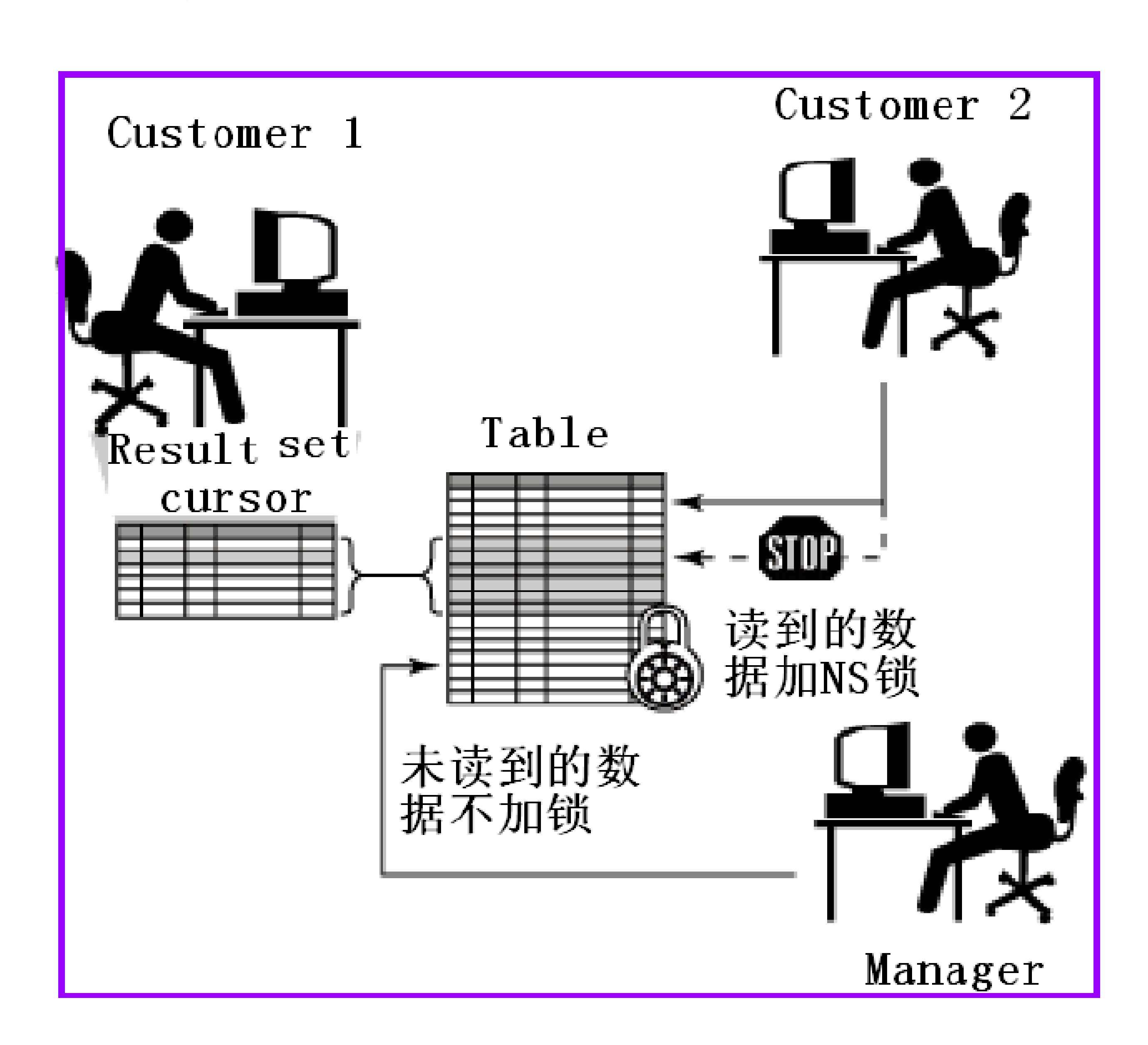


图 10-7　读稳定性隔离级别的示例

在使用读稳定性隔离级别时，一个事务中所有被读取过的行上都会被加上 NS 锁，直到该事务被提交或回滚，行上的锁才会被释放。这样可以保证在一个事务中即使多次读取同一行，得到的值也不会改变。但是，如果使用这种隔离级别，在一个事务中，如果使用同样的搜索标准重新打开已被处理过的游标，则结果集可能改变(可能会增加某些行，这些行被称为幻影行(Phantom)。这是因为 RS 隔离级不能阻止通过插入或更新操作在结果集中加入新行)。

读稳定性(RS)只锁定应用程序在工作单元中检索的那些行。它确保在某个工作单元完成之前，在该工作单元运行期间的任何限定行读取不被其他应用程序进程更改，且确保不会读取由另一个应用程序进程所更改的任何行，直至该进程落实了这些更改。也就是说，不可能出现“不可重复读”情形。

“读稳定性”隔离级别的其中一个目标是提供较高并行性以及数据的稳定视图。为了有助于达到此目标，优化器确保在发生锁定升级前不获取表级锁定。

“读稳定性”隔离级别最适用于包括下列所有特征的应用程序：

- 在并发环境下运行
- 须要限定某些行在工作单元运行期间保持稳定
- 在工作单元中不会多次发出相同的查询，或者在同一工作单元中发出多次查询时并不要求该查询获得相同的回答

10.3.3 游标稳定性(CS—Cursor Stability)

游标稳定性隔离级别在隔离事务效果方面非常宽松。它可以防止脏读，但有可能出现不可重复读和幻像读。这是因为在大多数情况下，游标稳定性隔离级别只锁定事务声明并打开的游标当前所引用的行。

当使用游标稳定性隔离级别的事务通过游标从表中检索行时，其他事务不能更新或删除游标所引用的行。但是，如果被锁定的行本身不是用索引访问的，那么其他事务可以将新的行添加到表中，以及对被锁定行前后的行进行更新和/或删除操作。所获取的锁一直有效，直到游标重定位或事务终止为止(如果游标重定位，原来行上的锁就被释放，并获得游标现在所引用行上的锁)。此外，如果事务修改了它检索到的任何行，那么在事务终止之前，其他事务不能更新或删除该行，即使游标不再位于被修改的行。与可重复读和读稳定性隔离级别一样，其他事务在其他行上进行的更改，在这些更改提交之前对于使用游标稳定性隔离级别(这是默认的隔离级别)的事务是不可见的。

如果如家酒店客房预定程序在游标稳定性隔离级别下运行，那么会有什么影响呢？当一个顾客检索某个日期段内所有可用房间的列表，然后查看产生的列表上每个房间的信息时(每次查看一个房间)，您可以更改旅馆中任何房间的房价，但是这个顾客当前正在查看的房间除外(对于指定的日期段)。同样，其他顾客可以对任何房间进行或取消预订，但是这个顾客当前正在查看的房间除外(对于指定的日期段)。也就是说，对于第一个顾客当前正在查看的房间记录，您和其他顾客都不能进行任何操作。当第一个顾客查看列表中另一个房间的信息时，您和其他顾客就可以修改他刚才查看的房间记录(如果这个顾客没有预订这个房间的话)。图 10-8 说明了这种行为。

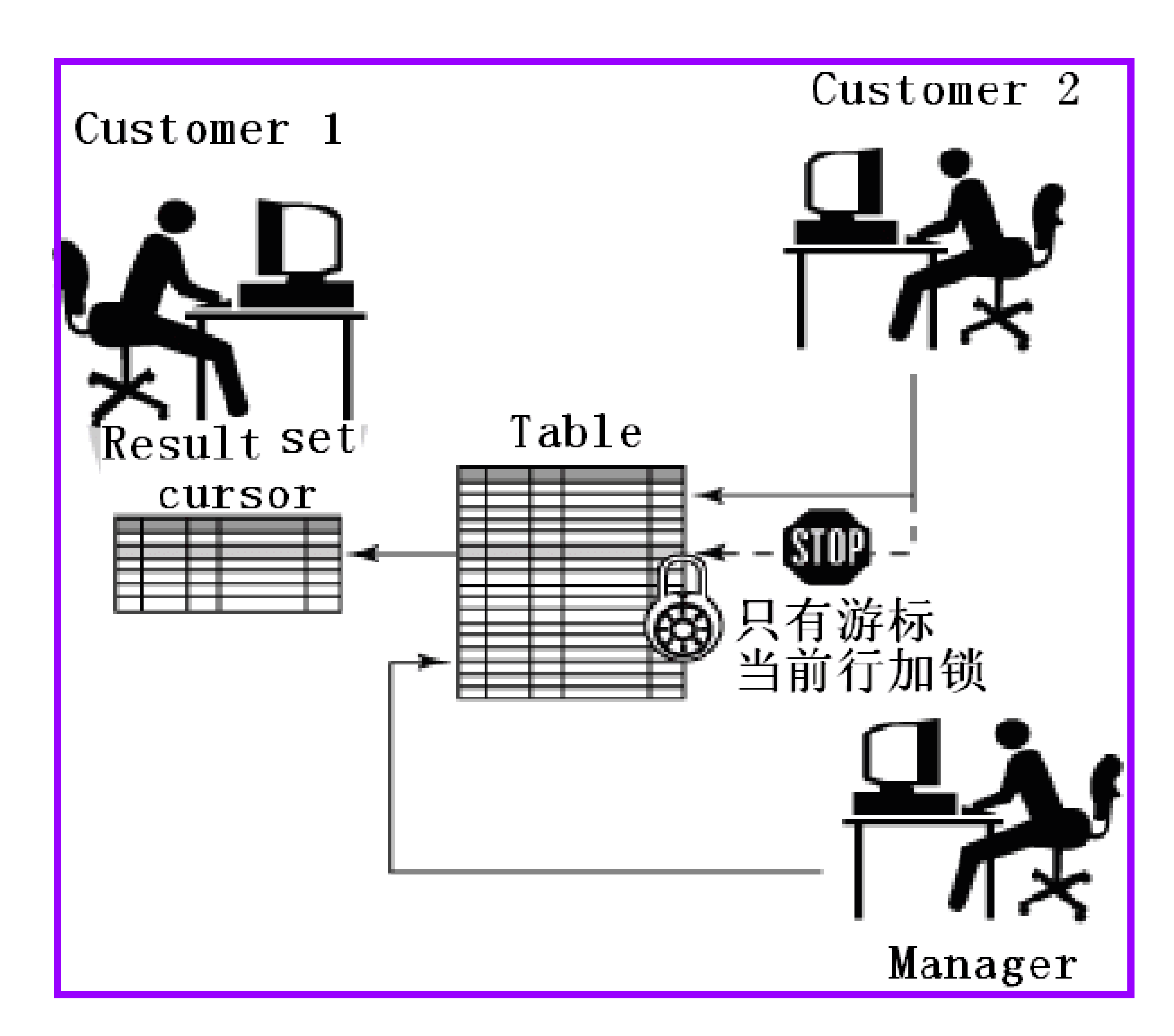


图 10-8　游标稳定性隔离级别的示例

在使用游标稳定性隔离级别时，一个事务的结果集中只有正在被读取的那一行(游标指向的行)会被加上 NS 锁，其他未被处理的行上不被加锁。这种隔离级别只能保证正在被处理的行的值不会被其他并发的程序所改变。该隔离级别是 DB2 默认的隔离级别。

当在行上定位游标时，游标稳定性(CS)会锁定任何由应用程序的事务所访问的行。此锁定在读取下一行或终止事务之前有效。但是，如果更改了某一行上的任何数据，那么在对数据库落实更改之前必须挂起该锁定。

对于具有“游标稳定性”的应用程序已检索的行，当该行上有任何可更新的游标时，任何其他应用程序都不能更新或删除该行。“游标稳定性”应用程序不能查看其他应用程序的未落实更改。

使用“游标稳定性”隔离级别，可能会出现不可重复读和幻像读现象。“游标稳定性”是默认隔离级别，建议在需要最大并行性，但只看到其他应用程序中已落实行的情况下才使用。

10.3.4　未提交读(UR—Uncommitted Read)

未提交读隔离级别是最不严格的隔离级别。实际上，在使用这个隔离级别时，仅当另一个事务试图删除或更改被检索的行所在的表时，才会锁定该检索的行。因为在使用这种隔离级别时，行通常保持未锁定状态，所以脏读、不可重复读和幻像读都可能会发生。因此，未提交读隔离级别通常用于那些访问只读表和视图的事务，以及某些执行 SELECT 语句的事务(只要其他事务的未提交数据对这些语句没有负面效果)。

顾名思义，其他事务对行所作的更改在被提交之前对于使用未提交读隔离级别的事务

是可见的。但是，此类事务不能看见或访问其他事务所创建的表、视图或索引，直到那些事务被提交为止。类似地，如果其他事务删除了现有的表、视图或索引，那么仅当进行删除操作的事务终止时，使用未提交读隔离级别的事务才能知道这些对象不再存在了(一定要注意一点：当运行在未提交读隔离级别下的事务使用可更新游标时，该事务的行为和在游标稳定性隔离级别下运行一样，并应用游标稳定性隔离级别的约束)。

那么未提交读隔离级别对如家酒店客房预定应用程序有什么影响呢？现在，当一个顾客检索某个日期段内的所有可用房间列表时，您可以更改旅馆中任何房间在任何日期的房价，而其他顾客也可以对任何房间进行或取消预订，包括第一个顾客当前正在查看的房间记录(对于指定的日期段)。另外，第一个顾客生成的房间列表可能包含其他顾客正在预订(因此实际上不可用)的房间。图 10-9 说明了这种行为。

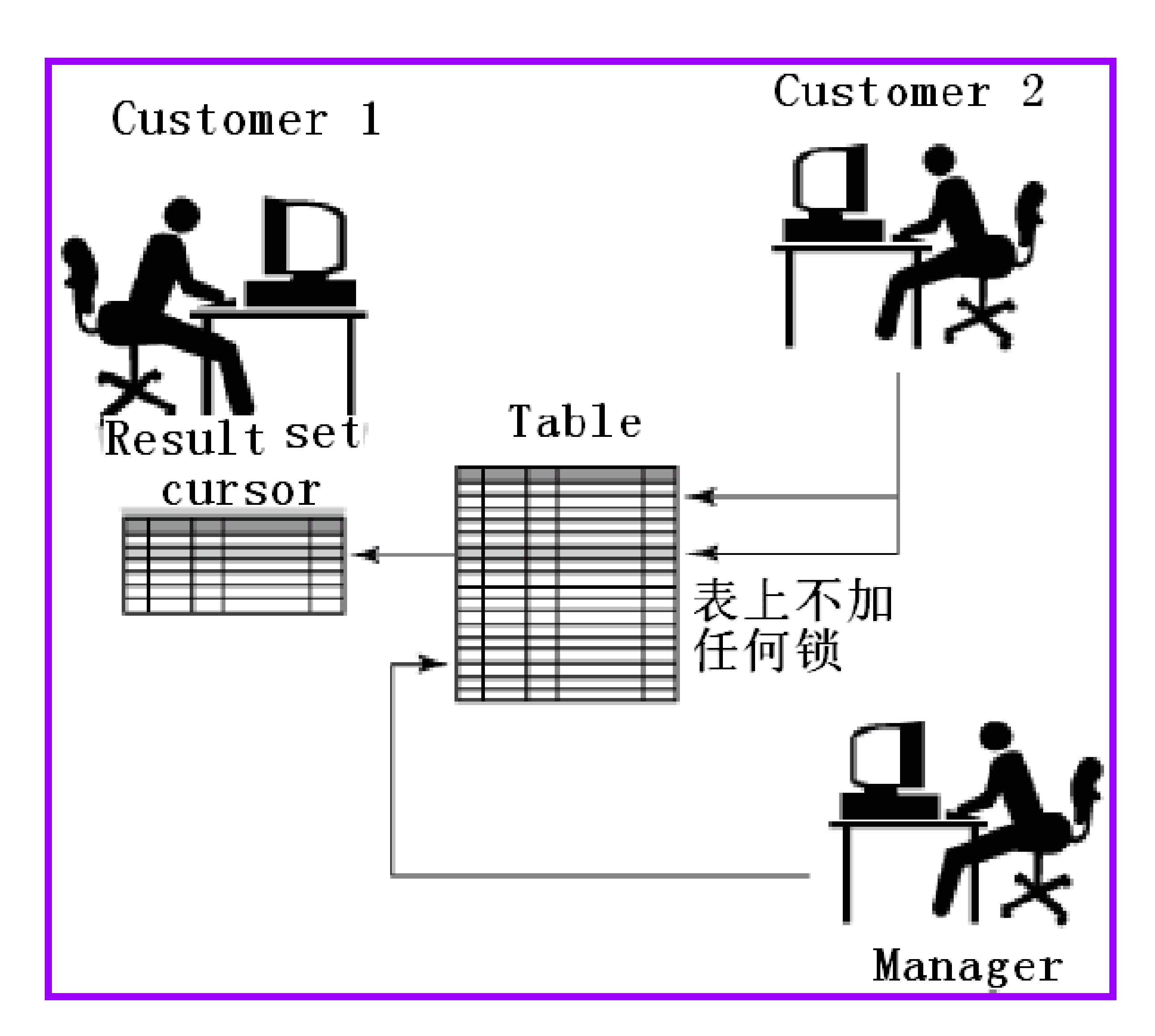


图 10-9　未提交读隔离级别示例

未提交读(UR)允许应用程序访问其他事务的未提交的更改。除非其他应用程序尝试删除或改变该表，否则该应用程序不会锁定正读取的行而使其他应用程序不能访问该行。对于只读和可更新的游标，“未提交的读”的工作方式有所不同。

在使用未提交读隔离级别时，对于只读操作，不加行锁。典型的只读操作包括：SELECT 语句的结果集(比如语句中包括 ORDER BY 子句)；定义游标时指明 FOR FETCH ONLY。

未提交读隔离级别可以改善应用程序的性能，同时可以最大程度地允许并发。但是，应用程序的数据完整性将受到威胁。如果需要读取未提交的数据，该隔离级是唯一选择。

只读游标可以访问大多数其他事务未落实的更改。但是，当该事务正在处理时，不能

使用正由其他事务创建或删除的表、视图和索引。其他事务的任何其他更改在落实或回滚前都可被读取。

注意：

“未提交读”隔离级别下的可更新操作的游标的工作方式游标稳定性隔离级别下的相同。

当使用隔离级别 UR 运行程序时，应用程序可以使用隔离级别 CS。发生这种情况的原因是应用程序中使用的游标是模糊游标。利用 BLOCKING 选项，可以将模糊游标升级为隔离级别 CS。BLOCKING 选项的默认值是 UNAMBIG。这意味着模糊游标是可更新的，并且隔离级别可以升级为 CS。要防止此升级，有两种选择：

- 修改应用程序中的游标为非模糊游标。将 SELECT 语句更改为包括 FOR READ ONLY 子句。
- 将模糊游标保留在应用程序中，但是使用预编译程序或 BLOCKING ALL 和 STATICREADONLY YES 选项绑定它，以允许在运行该程序时将任何模糊游标视为只读游标。

同对扫描 1000 行的“可重复读”给出的示例一样，如果使用“未提交读”，那么不需要任何行锁定。

使用“未提交读”，可能出现不可重复读和幻像读现象。“未提交读”隔离级别最常用于只读表上的查询。如果仅执行选择语句且不关心是否可从其他应用程序中看到未提交的数据时，那么该隔离级别也是不错的选择。

以上我们所讲的隔离级别的加锁范围和持续时间都是针对读操作而言的。对于更改操作，被修改的行上会被加上 X 锁，无论使用何种隔离级别，X 锁都直到提交或回滚之后才会被释放。

10.3.5 隔离级别的摘要

表 10-7 按不期望的结果概述了几个不同的隔离级别。

表 10-7 隔离级别摘要

隔 离 级 别	访问未落实的数据	不可重复读	幻像读现象
可重复读(RR)	不可能	不可能	不可能
读稳定性(RS)	不可能	不可能	可能
游标稳定性(CS)	不可能	可能	可能
未落实的读(UR)	可能	可能	可能

表 10-8 提供了简单的试探方法，以帮助您为应用程序选择初始隔离级别。首先考虑表 10-8 中所示的方法，并参阅先前对各隔离级别的讨论，以找到最适合的隔离级别。

表 10-8　选择隔离级别的准则

应用程序类型	需要高数据稳定性	不需要高数据稳定性
读写事务	RS	CS
只读事务	RR 或 RS	UR

为一个应用程序选择适当的隔离级别对于该应用程序避免无法容忍的现象很重要。因为获取和释放锁定所需的 CPU 和内存资源会随隔离级别的不同而不同，所以隔离级别不但影响应用程序之间的隔离程度，而且还影响个别应用程序的性能特征。潜在的锁等待情况也会随隔离级别的不同而不同。

因为隔离级别确定访问数据时如何锁定数据并使数据不受其他进程影响，所以您应该选择能平衡并行性和数据完整性需求的隔离级别。您指定的隔离级别在工作单元运行期间生效。

1. 选择正确的隔离级别

使用的隔离级别不仅影响数据库对并发性的支持程度，而且影响并发应用程序的性能。通常，使用的隔离级别越严格，并发性就越小，某些应用程序的性能还可能会越低，因为它们要等待资源上的锁被释放。那么，如何决定要使用哪种隔离级别呢？最好的方法是确定哪些现象是不可接受的，然后选择能够防止这些现象发生的隔离级别：

- 如果正在执行大型查询，而且不希望并发事务所作的修改会导致查询的多次运行而返回不同的结果，则使用可重复读隔离级别。
- 如果希望在应用程序之间获得一定的并发性，并且希望限定的行在事务执行期间保持稳定，则使用读稳定性隔离级别。
- 如果希望获得最大的并发性，同时不希望查询看到未提交的数据，则使用游标稳定性隔离级别。
- 如果正在只读的表/视图/数据库上执行查询，或者并不介意查询是否返回未提交的数据，则使用未提交读隔离级别。

2. 指定要使用的隔离级别

尽管隔离级别控制事务级上的行为，但实际上是在应用程序级被指定的：

- 对于嵌入式 SQL 应用程序，在预编译时或在将应用程序绑定到数据库(如果使用延

迟绑定)时指定隔离级别。在这种情况下，使用 PRECOMPILE 或 BIND 命令的 ISOLATION 选项来设置隔离级别。

- 对于开放数据库连接(Open Database Connectivity，ODBC)和调用级接口(Call Level Interface，CLI)应用程序，隔离级别是在应用程序运行时通过调用指定了 SQL_ATTR_TXN_ISOLATION 连接属性的 SQLSetConnectAttr()函数进行设置的(另外，也可以通过指定 db2cli.ini 配置文件中的 TXNISOLATION 关键字的值来设置 ODBC/CLI 应用程序的隔离级别。但是，这种方法不够灵活，不能像第一种方法那样为一个应用程序中的不同事务修改隔离级别)。
- 对于 Java 数据库连接(Java Database Connectivity，JDBC)和 SQLJ 应用程序，隔离级别是在应用程序运行时通过调用 DB2 的 java.sql 连接接口中的 setTransactionIsolation()方法设置的。

当没有使用这些方法显式指定应用程序的隔离级别时，默认使用游标稳定性隔离级别。这个默认设置适用于从命令行处理程序(CLP)执行的 DB2 命令、SQL 语句和脚本，以及嵌入式 SQL、ODBC/CLI、JDBC 和 SQLJ 应用程序。因此，也可以为从 CLP 执行的操作(以及传递给 DB2 CLP 进行处理的脚本)指定隔离级别。在这种情况下，隔离级别是通过在建立数据库连接之前在 CLP 中执行 CHANGE ISOLATION 命令设置的：

```
C:\pp>db2 change isolation to ur
DB21027E  当连接至数据库时未能更改隔离级别。
C:\pp>db2 connect reset
DB20000I  SQL 命令成功完成。
C:\pp>db2 change isolation to ur
DB21053W  当连接至不支持 UR 的数据库时，会发生自动升级。
DB20000I  CHANGE ISOLATION 命令成功完成。
```

在 DB2 UDB 7.1 及更高版本中，能够指定特定查询所用的隔离级别，方法是在 SELECT SQL 语句中加上 WITH [RR | RS | CS | UR]子句。使用这个子句的简单 SELECT 语句示例如下所示：

```
SELECT * FROM EMPLOYEE WHERE EMPID = '001' WITH RR
```

如果应用程序在大多数时候需要比较宽松的隔离级别(以支持最大的并发性)，但是对于其中的某些查询必须防止某些现象出现，那么这个子句就是帮助您实现目标的好方法。

10.4　锁转换、锁等待、锁升级和死锁

10.4.1　锁转换及调整案例

更改已挂起的锁定方式称为转换。当一个进程访问它已挂起锁定的数据对象，且访问

方式需要比已挂起的锁定更严格时，数据库管理器将该对象上现有的锁模式与被请求的锁模式进行比较，如果需要的锁模式更高，将进行锁转换。一个进程在任一时间对数据对象只能挂起一个锁定，尽管它可以通过查询间接地对同一数据对象多次请求锁定。在一个事务中，当一个对象需要不同的加锁模式时，对该对象上加上更高模式的锁，否则将保持现有的锁模式。锁模式由高到低按照以下顺序排列：

表锁：IN→IS→S→IX→U→X→Z

行锁：S→U→X

其中 S 锁与 IX 锁的转换比较特殊，当应用程序拥有一个表上的 S 锁并请求 IX 锁的时候，锁转换的结果为 SIX 锁。或者，当应用程序拥有一个表上的 IX 锁并请求 S 锁的时候，锁转换的结果也是 SIX 锁。

某些锁定方式仅适用于表，而有些锁定方式仅适用于行。对于行，如果需要 X 锁且挂起 S 或 U(更新)锁，那么进行转换。

但是，在锁定转换这一点上，IX(意向互斥)和 S(共享)锁是特殊情况。S 和 IX 锁定的严格程度相当，所以如果挂起了一个而请求另一个，那么结果转换为 SIX(带意向互斥的共享)锁定。如果所请求的方式更严格，那么所有其他的转换都将导致所请求的锁定方式变成挂起的锁定方式。

当查询更新行时，也可以发生双重转换。如果行是通过索引访问读取的且锁定为 S，那么包含该行的表具有覆盖意向锁定。但是，如果锁定类型是 IS 而不是 IX，且随后更改了行，那么表锁定将转换为 IX，而行锁定转换为 X。

注意：

在执行 SQL 时，锁定转换通常隐式地进行，由数据库自动完成而不需要用户参与。了解不同的查询以及表和索引的组合能获得的锁定种类，有助于设计和调整您的应用程序。

表 10-9 是一个锁转换的例子，这里假设当前应用程序的隔离级为 RR，IXCOL=?谓词采用的扫描方式为索引扫描。

表 10-9 锁转换示例

语　　句	语句所需的表锁	是否需要锁转换	结　果　锁	说　　明
SELECT … FROM A WHERE IXCOL=?	IS	不需要	IS	该事务中的第一个语句，不会发生锁转换
SELECT … FROM A	S	需要	S	S 锁的模式比 IS 锁高，需要锁转换

(续表)

语　　句	语句所需的表锁	是否需要锁转换	结　果　锁	说　　明
SELECT … FROM A WHERE IXCOL=?	IS	不需要	IS	IS 锁的模式比 S 锁低，所以不需要锁转换
UPDATE A SET… WHERE IXCOL=?	IX	需要	SIX	原先是 S 锁，现在又申请 IX 锁，锁转换的结果为 SIX 锁
COMMIT	不需要锁	不需要	所有锁被释放	在提交时锁被释放
SELECT … FROM A WHERE IXCOL=?	IS	不需要	IS	新事务的开始
LOCK TABLE A IN EXCLUSIVE MODE	X	需要	X	X 锁的模式比 IS 锁高，需要进行锁转换
UPDATE A SET… WHERE IXCOL=?	IX	不需要	X	IX 锁的模式比 X 锁低，不需要锁转换
ROLLBACK	不需要锁	不需要	所有锁被释放	在回滚时锁被释放

注意：

所有的锁在提交或回滚时都会被释放，除了定义游标时与指定为 WITH HOLD 的游标相关的表锁。

10.4.2　锁升级及调整案例

数据库管理器可以自动将锁定从行级别升级为表级别。对于分区表，数据库管理器可以自动将锁定从行或块级别升级为数据分区级别。*maxlocks* 数据库配置参数用于指定触发锁定升级的百分比。获取触发锁定升级的锁定的表可能不受影响。每个锁在内存中都需要一定的内存空间，为了减少锁需要的内存开销，DB2 提供了锁升级这一功能。锁升级是通过对表加上非意图性的表锁，同时释放行锁来减少锁的数目，从而达到减少锁需要的内存开销的目的的。锁升级由数据库管理器自动完成，数据库的配置参数锁列表页面数(LOCKLIST)和应用程序占有百分比(MAXLOCKS)直接影响锁升级的处理，如图 10-10 所示。

在图 10-10 中，上图说明了应用程序超出允许它具有的总的锁内存百分比(MAXLOCKS 值)的情况。因此，数据库执行锁升级，将应用程序的锁内存百分比降低到 MAXLOCKS 指定的百分比之下。下图说明了更为常见的情况：到达总的锁内存限制，因此执行锁升级来释放锁内存。

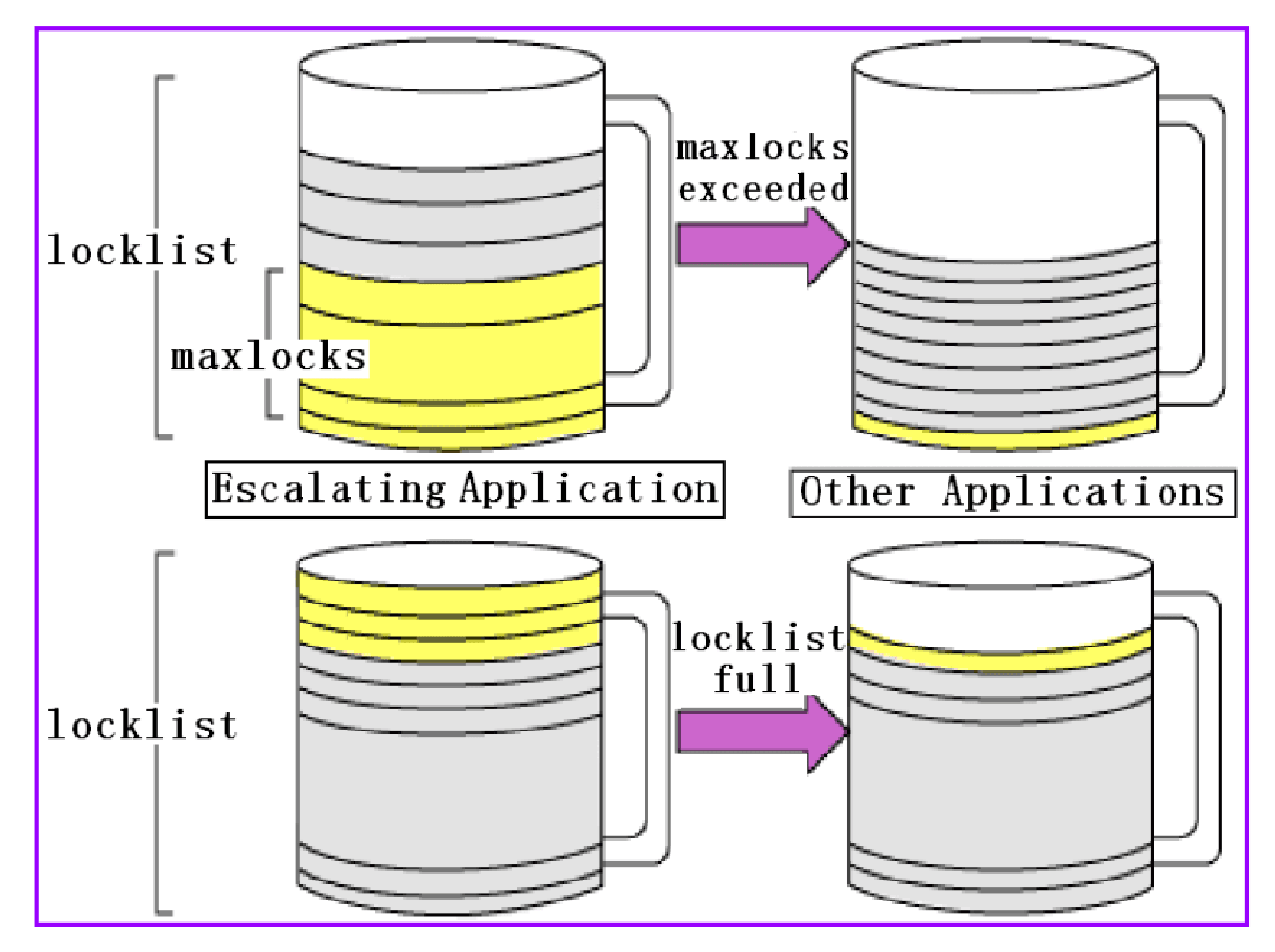


图 10-10　锁升级示意图

锁升级问题可以通过增加 LOCKLIST 和 MAXLOCKS 数据库参数的大小来解决。但是，如果仍然遇到锁定问题，应检查是否因未能提交事务而未释放已更新行上的锁。

每个数据库都有一个锁列表，该列表包含所有同时连接到数据库的应用程序所持有的锁。在 32 位平台上，一个对象上的第一个锁要求占 64 字节，而其他的锁要求占 32 字节。在 64 位平台上，第一个锁要求占 112 字节，而其他锁要求占 56 字节。

当一个应用程序使用的 LOCKLIST 的百分比达到 MAXLOCKS 时，数据库管理器将执行一次锁升级(lock escalation)，在这个操作中将使行锁转换成单独的一个表锁。而且，如果 LOCKLIST 快要耗尽，数据库管理器将找出持有一个表上最多行锁的连接，并将这些行锁转换成表锁，以释放 LOCKLIST 内存。锁定整个表会大大降低并发性，死锁的几率也就增加了。

- LOCKLIST 表明分配给锁列表的存储容量。每个数据库都有一个锁列表，锁列表包含了并发连接到该数据库的所有应用程序所持有的锁。锁定是数据库管理器用来控制多个应用程序并发访问数据库中数据的机制。行和表都可以被锁定。
- MAXLOCKS 定义了应用程序持有的锁列表的百分比，在数据库管理器执行锁升级之前必须填充该锁列表。当一个应用程序所使用的锁列表百分比达到 MAXLOCKS 时，数据库管理器会升级这些锁，这意味着用表锁代替行锁，从而减少列表中锁的数量。当任何一个应用程序所持有的锁数量达到整个锁列表大小的这个百分比时，对该应用程序所持有的锁进行锁升级。如果锁列表用完了空间，那么也会发

生锁升级。数据库管理器通过查看应用程序的锁列表并查找行锁最多的表，来决定对哪些锁进行升级。如果用一个表锁替换这些行锁，将不再会超出 MAXLOCKS 值，那么锁升级就会停止。否则，锁升级就会一直进行，直到所持有的锁列表百分比低于 MAXLOCKS。MAXLOCKS 参数乘以 MAXAPPLS 参数的值不能小于 100。

LOCKLIST 配置参数的计算方法如下(操作系统为 32 位平台)：

(1) 计算锁列表大小的下限：(512 * 32 * MAXAPPLS)/4096。其中，512 是每个应用程序平均所含锁数量的估计值，32 是对象(已有一把锁)上每把锁所需的字节数。

(2) 计算锁列表大小的上限：(512 * 64 * MAXAPPLS)/4096。其中，64 是某个对象上第一把锁所需的字节数。

(3) 对于您的数据，估计可能具有的并发数，并根据您的预计为锁列表选择一个初始值，该值位于您计算出的上限和下限之间。

MAXLOCKS 配置参数的计算方法如下：

MAXLOCKS = 100 * (512 锁/应用程序 * 32 字节/锁 * 2)/(LOCKLIST * 4096 字节)

该公式允许任何应用程序持有的锁是平均数的两倍。如果只有几个应用程序并发地运行，则可以增大 MAXLOCKS，因为在这些条件下锁列表空间中不会有太多争用。

锁升级会在以下两种情况下被触发：

- 某个应用程序请求的锁所占用的内存空间超出了 MAXLOCKS 和 LOCKLIST 的乘积大小。这时，数据库管理器将试图通过为提出锁请求的应用程序申请表锁，并释放行锁来节省空间。
- 在一个数据库中已被加上的全部锁所占的内存空间超出了 LOCKLIST 定义的大小。这时，数据库管理器也将试图通过为提出锁请求的应用程序申请表锁，并释放行锁来节省空间。

虽然升级过程本身并不用花很多时间，但是锁定整个表(相对于锁定个别行)降低了并发性，而且数据库的整体性能可能会由于对受锁升级影响的表的后续访问而降低。

在设计良好的数据库中，很少发生锁定升级。如果锁定升级将并行性降低到不可接受的程度(由 *lock_escalation* 监视元素监视)，那么就需要分析问题并决定如何解决此问题。

锁升级是有可能失败的，比如，现在一个应用程序已经在一个表上加有 IX 锁，表中的某些行上加有 X 锁，另一个应用程序又来请求表上的 IS 锁，以及很多行上的 S 锁，由于申请的锁数目过多引起锁的升级。数据库管理器试图为该应用程序申请表上的 S 锁来减少所需要的锁的数目，但 S 锁与表上原有的 IX 锁冲突，锁升级不能成功。

如果锁升级失败，引起锁升级的应用程序将接到一个－912 的 SQLCODE。

下面我们举一个实际的例子。

首先，运行下面的命令以打开针对锁的 DB2 监视器：

```
db2 -v update monitor switches using lock on
db2 -v terminate
```

然后收集数据库快照：

```
db2 -v get snapshot for database on sample | grep -i lock
```

在快照输出中，检查下列各项：

```
Locks held currently = 21224
Lock waits = 24683
Time database waited on locks (ms) = 32875
Lock list memory in use (Bytes) = 87224
Deadlocks detected = 89
Lock escalations = 8
Exclusive lock escalations = 12
Agents currently waiting on locks = 0
Lock Timeouts = 0
Internal rollbacks due to deadlock = 0
```

如果“Lock list memory in use (Bytes)”超过定义的 LOCKLIST 大小的 50%，那么就增加 LOCKLIST 数据库配置参数中的 4KB 页的数量。

如果发生了“Lock escalations>0”或“Exclusive lock escalations>0”，则应该或者增大 LOCKLIST 或者 MAXLOCKS，抑或同时增大两者。查看“Locks held currently”、“Lock waits”、“Time database waited on locks (ms)”、“Agents currently waiting on locks”和“Deadlocks detected”中是否存在高值，如果有的话，就可能是差于最优访问计划、事务时间较长或者应用程序并发问题的症状。

10.4.3 锁等待及调整案例

当应用程序不能够立刻得到对一个对象请求的锁时，该应用程序将进入一个等待服务的队列，等待占用该锁的应用程序提交或回滚来释放该锁，这种情况称为锁等待。锁定超时检测是一个数据库管理器功能，用于防止应用程序在异常情况下无限时地等待释放锁定。例如，一个事务可能正等待另一个用户的应用程序挂起的锁定，但该用户已离开工作站，而没有允许应用程序落实释放该锁定的事务。要避免在这种情况下应用程序继续等待，将 LOCKTIMEOUT 配置参数设置为任何应用程序应等到获取锁定的最长时间。这可以帮助避免全局死锁的情况发生。如果锁定请求处于暂挂的时间大于 LOCKTIMEOUT 值，那么请求应用程序将接收到一个错误并将其事务回滚。LOCKTIMEOUT 的默认值为 - 1，它关闭锁定超时检测，如果出现锁等待，应用程序将会出现无穷等待现象。例如，如果 APPL1 尝试获取一个已由 APPL2 挂起的锁定，那么 APPL1 在超时周期到期时返回

SQLCODE－911 (SQLSTATE 40001)，原因码为 68。对于生产系统中的 OLAP，LOCKTIMEOUT 一开始为 60 (秒)比较好，对于 OLTP 大约为 10 秒比较好。对于开发环境，应该使用－1，以识别和解决锁等待的情况。如果有大量的并发用户，可能需要增加 OLTP 时间，以避免回滚。

如果快照监控结果输出中“Lock Timeouts”是一个较高的数，那么可能是由以下原因造成的：

- LOCKTIMEOUT 的值太低
- 某个事务持有锁的时间有所延长
- 锁升级

锁等待可能会造成如下几种结果：

- 引起死锁(死锁是锁等待的一个特例)，由死锁检测器处理。
- 等待超时，等待的应用程序返回 SQLCODE –911 和原因代码 68，并自动回滚。超时的时间由数据库配置参数 LOCKTIMEOUT 设置，单位为秒。比如 LOCKTIMEOUT 的值为 60，则如果进行锁等待的应用程序在 60 秒后还不能得到所需的锁，该程序将自动回滚。
- 如果 LOCKTIMEOUT 的值被设置为－1，则应用程序永远等待，直到能够获得所需要的锁。
- 对于表、行、数据分区和 MDC 块锁定，应用程序可使用 SET CURRENT LOCK TIMEOUT 来覆盖数据库级别的 LOCKTIMEOUT 设置。

有时候，锁等待情形会导致锁超时，而锁超时又会导致事务被回滚。锁等待导致锁超时所需的时间段由数据库配置参数 LOCKTIMEOUT 指定。锁超时分析最大的问题是，不知道下一次的锁超时何时发生。为了捕捉死锁，可以创建一个死锁事件监视器。每当出现死锁时，这个死锁事件监视器便写一个条目。但是，对于锁超时就没有类似的事件监视器。下面我们来举一个例子说明如何发现锁超时。

现在的情况是，一个应用系统中出现很多锁等待导致系统出现了性能问题。

首先打开数据库监控开关：

```
    db2  update dbm cfg using DFT_MON_LOCK on DFT_MON_STMT on
    db2  update monitor switches using lock ON sort ON bufferpool ON uow ON table
ON statement ON
```

利用 DB2 表函数编写一个监控 SQL 脚本：

```
    select AGENT_ID ,substr(STMT_TEXT,1,100) as statement,STMT_ELAPSED_TIME_MS
from table(SNAPSHOT STATEMENT('sample',-1)) asBwhere AGENT ID in (select
AGENT_ID_HOLDING_LK from table(SNAPSHOT_LOCKWAIT('sample',-1)) as A  order by
```

```
LOCK_WAIT_START_TIME ASC FETCH FIRST 20 ROWS ONLY ) order by STMT_ELAPSED_TIME_MS
DESC
```

此语句会按照执行时间长短的先后顺序排列出所有造成 lockwait 的 SQL 语句。

我们可以把这个脚本写到一个 shell 脚本中，每隔三秒钟运行一次，把监控结果管道累加输出到一个文件：

```
    #!/usr/bin/ksh
    #
    dbname=$1
    #create a log file
    filename=find.locksql.$(date+'%m%d%H%M%S')
    touch $filename
    #connect to database
    echo now,connecting to database: $dbname
    db2 "connect to $dbname"
    db2 "update dbm cfg using DFT_MON_LOCK on DFT_MON_STMT on"
    db2  "update monitor switches using lock ON sort ON bufferpool ON uow ON table
ON statement ON"
    echo now,finding the SQLs which made lockwait
    db2  "select AGENT_ID ,substr(STMT_TEXT,1,100) as
statement,STMT ELAPSED TIME MS  from table(SNAPSHOT STATEMENT('$dbname',-1))
asBwhere AGENT_ID in (select AGENT_ID_HOLDING_LK from
table(SNAPSHOT LOCKWAIT('$dbname',-1)) as A  order by  LOCK WAIT START TIME
ASC FETCH FIRST 20 ROWS ONLY ) order by STMT_ELAPSED_TIME_MS DESC" > $filename
    echo The SQLs have saved to the file $filename
```

一旦定位了引起锁等待的 SQL 语句后，如果该 SQL 语句写的效率很低下，可以考虑对该 SQL 语句作出调整；如果该 SQL 语句中没有创建最合理的索引，可以考虑用 db2advis 工具为该 SQL 语句创建最合理的索引。

传统的锁定方法会导致应用程序互相阻塞。当一个应用程序必须等待另一个应用程序释放其锁定时，阻塞就会发生。用于处理这种阻塞的影响的策略通常会提供一种机制以指定最大可接受的阻塞持续时间。这就是应用程序在不能获取锁定的情况下在返回之前等待的时间。

以前，只能在数据库级别通过更改 LOCKTIMEOUT 数据库配置参数的值来指定时间。现在，锁定等待方式策略通过新的 SET CURRENT LOCK TIMEOUT 语句指定(DB2 V9.5 以后)，此语句更改 CURRENT LOCK TIMEOUT 专用寄存器的值。CURRENT LOCK TIMEOUT 专用寄存器指定在返回指示不能获取锁定错误之前等待锁定的秒数。

10.4.4　死锁及调整案例

死锁的产生是由于锁请求双方都彼此持有对方所需要的锁，这种情况下又去请求锁而导致死锁的产生。

下面我们通过一个典型的例子来说明死锁的概念。

假定事务 1 在表 A 上获取了互斥(X)锁，而事务 2 在表 B 上获取了互斥(X)锁。现在，假定事务 1 尝试在表 B 上获取互斥(X)锁，而事务 2 尝试在表 A 上获取互斥(X)锁。这两个事务的处理都将被挂起，直到同意第二个锁请求为止。但是，因为在任何一个事务释放它目前持有的锁(通过执行或回滚操作)之前，这两个事务的锁请求都不会被同意，而且因为这两个事务都不能释放各自目前持有的锁(因为它们都已挂起并等待锁)，所以它们都陷入了死锁循环。图 10-11 说明了这个死锁场景。

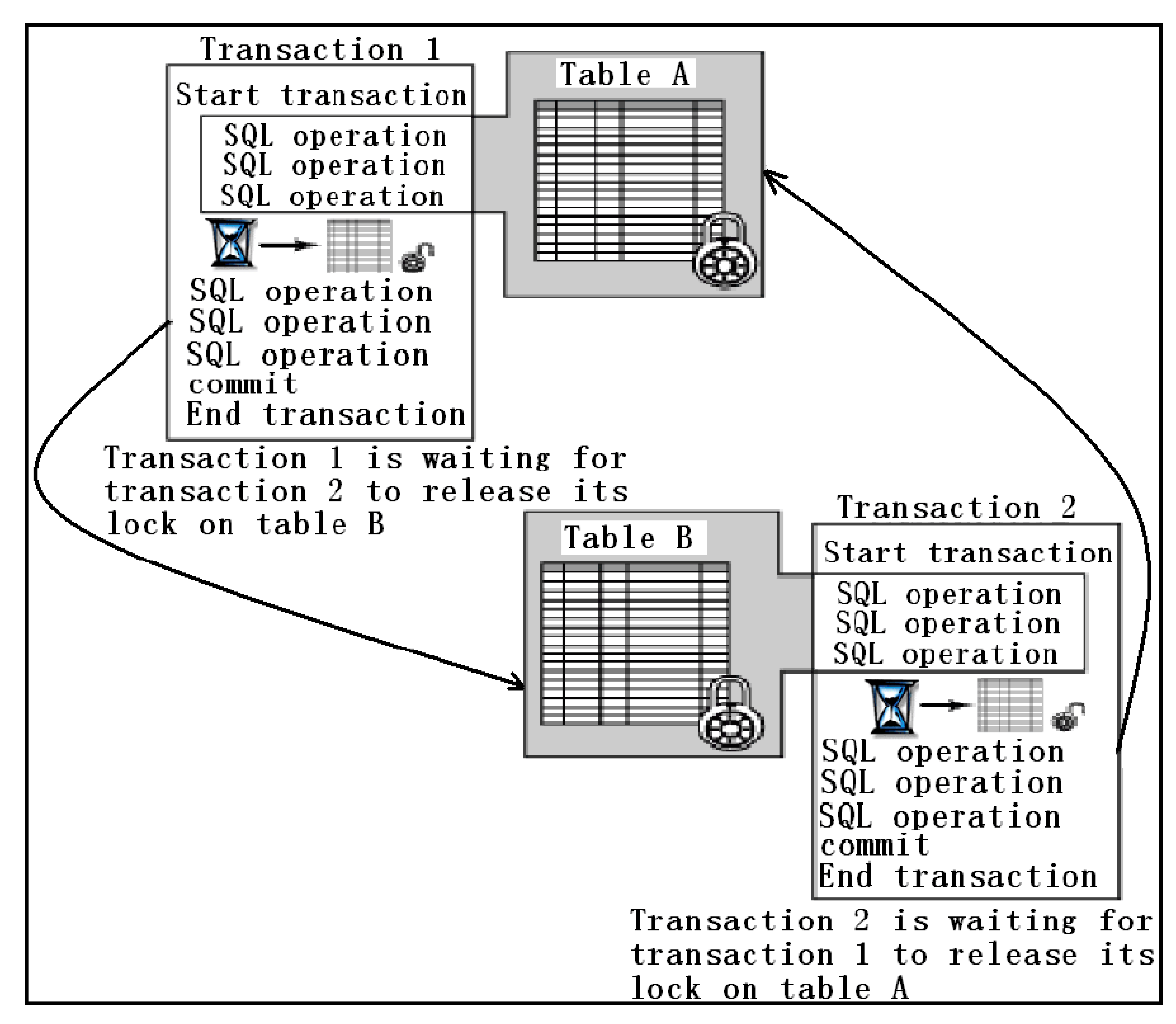


图 10-11　死锁循环示例

当死锁循环发生时，除非某些外部代理进行干涉，否则所涉及的所有事务将无限期地等待释放锁。在 DB2 UDB 中，用于处理死锁的代理是被称为死锁检测器的异步系统后台进程。死锁检测器的唯一职责是定位和解决在锁定子系统中找到的任何死锁。每个数据库都有自己的死锁检测器，它在数据库初始化过程中激活。激活之后，死锁检测器在大多数

时间处于“休眠”状态，但会以预置的时间间隔被“唤醒”，以确定锁定子系统中是否存在死锁循环。如果死锁检测器在锁定子系统中发现死锁，则随机选择死锁涉及的一个事务，终止并回滚它。选择的事务收到一个 SQL 错误编码(－911)，它所获得的所有锁都被释放。这样，剩下的事务就可以继续执行了，因为死锁循环已经被打破了。

这就是死锁的典型情况，两个应用程序互相持有对方所需要的锁，在得不到自己所需要的锁的情况下，也不会释放现有的锁。

下面我们再举一个死锁的例子。

模拟一个死锁场景。打开两个单独的命令行 CLP 窗口。

(1) 在第一个窗口中(图 10-12 所示)，执行以下命令(在每个命令后面按回车键)：

```
UPDATE COMMAND OPTIONS USING c OFF
(注意：这会关闭自动提交。)
CONNECT TO SAMPLE
CREATE TABLE deadtable (c1 INTEGER)
COMMIT
INSERT INTO deadtable VALUES (1)
```

```
db2 => CREATE TABLE deadtable (c1 INTEGER)
DB20000I  SQL命令成功完成。
db2 => COMMIT
DB20000I  SQL命令成功完成。
db2 => INSERT INTO deadtable VALUES (1)
DB20000I  SQL命令成功完成。
db2 =>

微软拼音 半:
```

图 10-12　死锁场景 1

(2) 在第二个窗口中(图 10-13 所示)，执行以下命令：

```
UPDATE COMMAND OPTIONS USING c OFF
(注意：这会关闭自动提交。)
CONNECT TO SAMPLE
INSERT INTO deadtable VALUES(2)
SELECT * FROM deadtable
(您会看到光标不动了。)
```

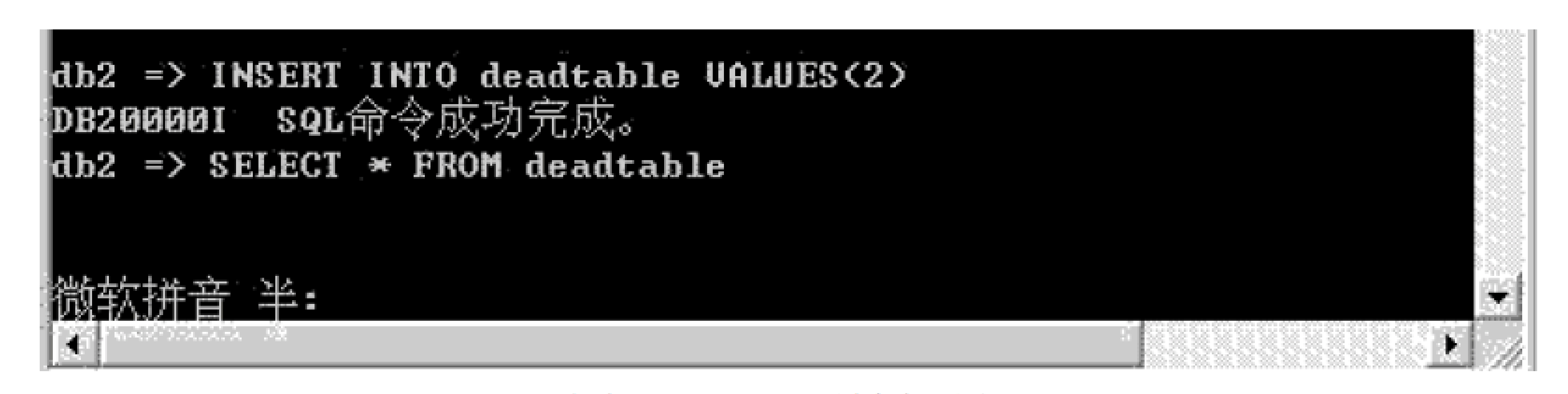


图 10-13　死锁场景 2

(3) 在完成第(2)步之后，等待大约 15 秒。然后，在第一个窗口中，执行以下命令(图 10-14 所示)：

```
SELECT * FROM deadtable
```



图 10-14　在第一个窗口中执行命令

(4) 这时会发生死锁，因为第一个窗口和第二个窗口都在等待对方释放锁，必须让其中一个应用程序回滚，才能打破锁冲突。DB2 死锁监视器进程会在 10 秒内选择一个应用程序并使它回滚。10 秒是默认设置，您可以通过数据库配置参数 DLCHKTIME 来设置。一个窗口将返回查询结果，另一个窗口将返回死锁消息(图 10-15 所示)。



图 10-15　第二窗口返回死锁信息

DB2 利用一个被称为死锁检测器的后台进程进行死锁的检测，该进程每隔一定的时间段进行一次检测，一旦发现死锁，该进程会选择一个牺牲者。牺牲者将自动回滚，释放掉占用的锁并返回 SQLCODE –911 和原因代码 2，死锁可以消除。

注意，DB2 deadlock detector(死锁检测器)会随机选择受害者，因此，不能确定哪一个更新将发生回滚。在上面描述的场景中，第二个更新发生回滚，但是您测试时，也许是第一个更新发生回滚。

dlchktime 是设置死锁检查间隔的配置参数，该参数以毫秒为单位，其有效范围是 1000 至 600000ms。该值过高将增加应用程序等待死锁被发现的时间，如果过低，死锁检测的间

隔虽然加快，但却降低了少许运行性能。该参数的默认值为 10000ms(10 秒)。

可能引起死锁的情况及其影响程度包括：

- 应用程序隔离级别采用重复读和读可靠性(主要)
- 锁升级(主要)
- 锁转换(次要)
- 编目表的更改(中等)
- 参照完整性的约束(次要)

确定并查看快照输出结果中“Deadlocks detected”中是否存在高值，如果有的话，就可能是差于最优访问计划、事务时间较长或者应用程序并发问题的症状。如果要发现死锁，那么需要创建一个针对死锁的事件监视器或 db2pd 工具。事件监视器带有详细信息，以便查看当前正在发生的事情。

```
create event monitor deadlock for deadlocks write to table
```

当我们在事件监视结果中定位了引起死锁的 SQL 语句后，如果该 SQL 语句写的效率很低，可以考虑对该 SQL 语句进行调整；如果该 SQL 语句中没有创建最合理的索引，可以考虑用 db2advis 工具为该 SQL 语句创建最合理的索引。

最小化死锁建议

- 在整个应用程序中，总是按相同次序访问资源可以最小化死锁。例如，如果一个应用程序组件将要访问表 A，然后是表 B，接着是表 C；而另一个应用程序组件需要访问表 A 和 C，那么第 2 个组件应该遵循先 A 后 C 的访问次序。
- 对于 DB2 V9 以前的版本，导致死锁的一个常见原因是锁列表数据库配置参数的大小不足，尤其是使用默认值时。因此应该合理地设置 LOCKLIST 和 MAXLOCKS 配置参数。默认情况下 DB2 V9 使用了 STMM，它会调整锁列表大小以避免可能由此引起的锁升级和死锁。
- 确保参照完整性(Referential Integrity，RI)关系中的依赖表拥有与外键匹配的索引。

10.5 锁相关的性能问题总结

调整锁定以实现并行性和数据完整性时，应考虑下列准则：

- 频繁使用 COMMIT 语句来创建较小的工作单元以使许多用户可以并发访问数据。当应用程序在逻辑上一致时，即当更改的数据一致时，发出 COMMIT 语句。当发出 COMMIT 时，释放锁定，与声明了 WITH HOLD 的游标相关的表锁定除外。

- 在发出 COMMIT 语句之前，应先关闭 CURSOR WITH HOLD。在某些情况下，在结果集关闭并且事务落实后锁定仍然存在。在发出 COMMIT 语句之前关闭 CURSOR WITH HOLD 可确保释放锁定。
- 指定适当的隔离级别。即使应用程序很少读取行，也会获得锁定，所以落实只读工作单元仍很重要。这是因为在只读应用程序中通过可重复读、读稳定性及游标稳定性隔离级别可以获得共享锁定。借助于可重复读和读稳定性，可挂起所有锁定，直到发出 COMMIT，这可以阻止其他进程更新锁定的数据，除非您使用 WITH RELEASE 子句关闭您的游标。此外，甚至在使用动态 SQL 语句的未落实的读应用程序中也要获得目录锁定。数据库管理器确保应用程序不检索未落实的数据(其他应用程序已经更新但尚未落实的行)，除非正在使用未落实的读隔离级别。
- 适当地使用 LOCK TABLE 语句。该语句锁定整个表。但只锁定 LOCK TABLE 语句中指定的表。不锁定指定表的父表和从属表。必须确定是否需要锁定其他可访问的表以便在并行性与性能方面达到期望的结果。在工作单元被落实或回滚之前不释放锁定。

以共享方式锁定表

想要访问在时间上一致的数据，即表在特定时间点的最新数据。如果表活动频繁，那么确保整个表保持稳定的唯一方法是将其锁定。例如，应用程序想要抽取表的快照。但是，在应用程序需要处理表的一些行期间，其他应用程序正在更新还没有处理的行。可重复读允许这样的操作，但您不希望这样。

另一种方法是，应用程序可以发出 LOCK TABLE IN SHARE MODE 语句：无论是否检索到行，都不能更改任何行。然后可以按需要检索任意多行，应了解已经检索的行就在检索它们之前还没有被更改。

使用 LOCK TABLE IN SHARE MODE，其他用户可从表中检索数据，但不能更新、删除表中的行，或将行插入表中。

以互斥方式锁定表

想要更新表的大部分。相对于更新表时锁定每一行，然后在落实所有更改时解锁行而言，该方式阻止所有其他用户访问该表的开销少并且更有效。

使用 LOCK TABLE IN EXCLUSIVE MODE，所有其他用户都被锁定在外；没有其他应用程序可以访问该表，除非它们是未落实的读应用程序。

- 在应用程序中使用 ALTER TABLE 语句。

带 LOCKSIZE 参数的 ALTER TABLE 语句是 LOCK TABLE 语句的备用语句。LOCKSIZE 参数允许您指定下一次表访问的 ROW 锁定或 TABLE 锁定的锁定详细程度。

当新建一个表时，选择 ROW 锁定同选择默认锁定大小无任何区别。选择 TABLE 锁定可提高查询性能，方法是限制需要获取的锁定的数目。但是，并行性可能由于所有锁定位于完整的表上而降低。仍然会对所有其他操作执行行级别锁定，并对键插入执行行级别锁定以保护可重复读(RR)扫描程序。

- 关闭游标以释放它们挂起的锁定。

当使用包括 WITH RELEASE 子句的 CLOSE CURSOR 语句关闭游标时，数据库管理器尝试释放所有为游标挂起的读锁定。表读锁定是 IS、S 和 U 表锁定。行读锁定是 S、NS 和 U 行锁定。块读锁定是 IS、S 和 U 块锁定。

WITH RELEASE 子句对正在 CS 或 UR 隔离级别下操作的游标不起作用。当对正在 RS 或 RR 隔离级别下操作的游标指定 WITH RELEASE 子句后，它将结束那些隔离级别的一些保证。明确地说，RS 游标可能经历不可重复读现象，而 RR 游标不会发生不可重复读或幻像读现象。

如果一个原来是 RR 或 RS 的游标通过使用 WITH RELEASE 子句关闭后重新打开，那么将获得新的读锁定。

在 CLI 应用程序中，DB2 CLI 连接属性 SQL_ATTR_CLOSE_BEHAVIOR 可用来获得与 CLOSE CURSOR WITH RELEASE 相同的结果。

10.6 锁与应用程序设计

为了确定锁定属性，可以将应用程序处理划分为下列类型的其中一种：

- 只读

此类型包括所有这样的选择语句，它们本身是只读的，并具有一个显式的 FOR READ ONLY 子句；或者是模糊的(但是查询编译器因为 PREP 或 BIND 命令指定了 BLOCKING 选项的值而假定它们是只读的)。此处理类型只要求“共享”锁定(S、NS 或 IS)。

- 更改意向

此类型包括具有 FOR UPDATE 的 SELECT 语句，或者查询编译器解释有歧义的语句以表示想进行更改。此类型使用“共享”和“更新”锁定(对于行，为 S、U 和 X；对于表，为 IX、U 和 X)。

- 更改

此类型包括 UPDATE、INSERT 及 DELETE，但不包括 UPDATE WHERE CURRENT OF 或 DELETE WHERE CURRENT OF。此类型要求“互斥”锁定(X 或 IX)。

- 受控游标

此类型包括 UPDATE WHERE CURRENT OF 和 DELETE WHERE CURRENT OF。此

类型也要求“互斥”锁定(X 或 IX)。

根据子查询语句的结果，在目标表中进行插入、更新或删除数据的语句，执行两种类型的处理：只读处理规则确定对在子选择语句中返回的表的锁定；更改处理规则确定对目标表的锁定。

优化器假定应用程序必须检索由 SELECT 语句指定的所有行，此假设最适合于 OLTP 和批处理环境。但是，在“浏览”应用程序中，查询经常定义一个可能很大的答案集，但它们只检索前几行，通常只检索填满该屏幕所需的那么多行。

要提高这种应用程序的性能，可以按下列方式修改 SELECT 语句：

- 使用 FOR UPDATE 子句来指定可由后续定位的 UPDATE 语句更新的列；
- 使用 FOR READ/FETCH ONLY 子句来使返回的列为只读的；
- 使用 OPTIMIZE FOR *n* ROWS 子句来授予检索整个结果集中前 *n* 行的优先级；
- 使用 FETCH FIRST *n* ROWS ONLY 子句来仅检索指定的几行；
- 使用 DECLARE CURSOR WITH HOLD 语句来每次检索一行。

注意：

如果使用 FOR UPDATE、FETCH FIRST n ROWS ONLY 或 OPTIMIZE FOR n ROWS 子句，或者您声明游标为“滚动”，那么会影响行分块。

FOR UPDATE 子句

FOR UPDATE 子句通过仅包含可由后续定位的 UPDATE 语句更新的列来限制结果集。如果不带列名指定 FOR UPDATE 子句，那么包括表或视图的全部可更新列。如果指定列名，那么每个名称必须是非限定的，且必须标识表或视图的某一列。

在下列情况下，不能使用 FOR UPDATE 子句：

- 不能删除与 SELECT 语句关联的游标；
- 选择的列中至少有一列是目录表的不可更新列，且没有在 FOR UPDATE 子句中排除掉。

由于相同的目的，可将 DB2 CLI 连接属性 SQL_ATTR_ACCESS_MODE 用于 CLI 应用程序中。

FOR READ 或 FETCH ONLY 子句

FOR READ ONLY 子句或 FOR FETCH ONLY 子句确保返回只读结果。因为被定义为只读的视图上 SELECT 的结果表也是只读的，所以允许此子句但没有作用。

对于允许更新和删除的结果表，如果数据库管理器可以检索数据块而不是互斥锁定，那么指定 FOR READ ONLY 可能会提高 FETCH 操作的性能。不要对用于定位的 UPDATE

或 DELETE 语句的查询使用 FOR READ ONLY 子句。

由于相同的目的，可将 DB2 CLI 连接属性 SQL_ATTR_ACCESS_MODE 用于 CLI 应用程序。

OPTIMIZE FOR n ROWS 子句

OPTIMIZE FOR 子句声明它只想检索结果的一个子集或优先检索前几行。优化器然后可以优先选择把检索前几行的响应时间减至最短的访问方案。另外，作为单个块发送到客户机的行数由 OPTIMIZE FOR 子句中的“n”值来限制。因此，OPTIMIZE FOR 子句既影响服务器从数据库检索合格行的方式，又影响将合格行返回到客户机的方式。

例如，假设您定期查询职员表，来查找具有最高工资的职员。

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
        FROM EMPLOYEE        ORDER BY SALARY DESC
```

您定义了一个基于 SALARY 列的降序索引。但是，由于职员是按职员号排序的，所以工资索引可能很难集群。为了尽量避免许多随机的同步 I/O，优化器将可能选择使用列表预取访问方法，此方法需要对合格的所有行的行标识排序。在将前几个合格行返回至应用程序前，此排序可能导致一个延迟。要防止此延迟，向语句添加 OPTIMIZE FOR 子句，如下所示：

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY FROM EMPLOYEE
          ORDER BY SALARY DESC        OPTIMIZE FOR 20 ROWS
```

在此情况下，优化器可能选择直接使用 SALARY 索引，因为只检索具有最高工资的 20 个职员。不管可以将多少行分块，而只把每 20 行组成的一个块返回至客户机。

使用 OPTIMIZE FOR 子句，优化器优先选择可以避免大量操作或中断行流动(如排序)的访问方案。使用 OPTIMIZE FOR 1 ROW 最有可能影响访问路径。使用此子句可以有下列作用：

- 连接方法可以更改。一个嵌套循环连接是非常可能的选择，因为它具有低开销成本，并且通常在检索少量行时更有效率。
- 一个与 ORDER BY 子句匹配的索引更可能，因为对于 ORDER BY 不需要排序。
- 列表预取不太可能，因为此访问方法需要排序。
- 顺序预取不太可能，因为只需要知道少量的几行。
- 在一个连接查询中，在 ORDER BY 子句中包含列的表可能被选作外部表，前提是该外部表上的一个索引提供 ORDER BY 子句所需的排序。

虽然 OPTIMIZE FOR 子句适用于所有优化级别，但它在优化级别 3 和更高级别下工作得最好。因为级别 3 以下的级别使用“贪婪”连接枚举方法，此方法有时会产生一个不能使它们自己很快检索前几行的多表连接的访问方案。

OPTIMIZE FOR 子句不阻止您检索全部合格行。如果确实要检索全部合格行，那么总耗用时间可能大大高于优化器为整个答案集进行优化所需的时间。

如果已打包的应用程序使用调用级接口(DB2 CLI 或 ODBC)，可在 db2cli.ini 配置文件中使用 OPTIMIZE FOR N ROWS 关键字，让 DB2 CLI 自动将一个 OPTIMIZE FOR n ROWS 子句追加至每个查询语句的末尾。

FETCH FIRST n ROWS ONLY 子句

FETCH FIRST *n* ROWS ONLY 子句设置可检索的最大行数。将结果表限制为只包含前几行可提高性能。无论结果集可能另外包含多少行，只检索 *n* 行。

如果同时指定了 FETCH FIRST 子句和 OPTIMIZE FOR 子句，那么这两个值中较小的一个影响通信缓冲区(RQRIOBLK，最大请求 I/O 块)大小。为了达到最优化，将这两个值看做是互不相关的。

DECLARE CURSOR WITH HOLD 语句

当用包括 WITH HOLD 子句的 DECLARE CURSOR 语句声明游标时，在落实该事务时任何打开的游标仍然打开，并且释放所有锁定(保护打开的 WITH HOLD 游标的当前游标位置的锁定除外)。

如果回滚事务，那么关闭所有打开的游标并释放所有锁定。

10.7　锁监控工具

在 DB2 中对锁进行监控有很多工具：快照监控、事件监控和 db2pd。下面我们分别举例。

1. 快照监控方式

在使用快照方式对锁进行的监控前，必须把监控锁的开关打开，可以从实例级别或会话级别打开，具体命令如下：

```
db2 update dbm cfg using dft_mon_lock on --实例级别

db2 update monitor switches using lock on  --会话级别，推荐使用
```

开关打开后，可以执行下列命令来进行锁的监控：

```
db2 get snapshot for locks on dev
```

数据库 dev 中具体锁的详细信息如下：

```
                Database Lock Snapshot
Database name                              = DEV
Database path                              = /db2/DEV/db2dev/NODE0000/SQL00001/
Input database alias                       = DEV
Locks held                                 = 49
Applications currently connected           = 38
Agents currently waiting on locks          = 6
Snapshot timestamp                         = 08-15-2003 15:26:00.951134
Application handle                         = 6
Application ID                             = *LOCAL.db2dev.030815021007
Sequence number                            = 0001
Application name                           = disp+work
Authorization ID                           = SAPR3
Application status                         = UOW Waiting
Status change time                         =
Application code page                      = 819
Locks held                                 = 0
Total wait time (ms)                       = 0
Application handle                         = 97
Application ID                             = *LOCAL.db2dev.030815060819
Sequence number                            = 0001
Application name                           = tp
Authorization ID                           = SAPR3
Application status                         = Lock-wait
Status change time                         = 08-15-2003 15:08:20.302352
Application code page                      = 819
Locks held                                 = 6
Total wait time (ms)                       = 1060648
 Subsection waiting for lock               = 0
  ID of agent holding lock                 = 100
  Application ID holding lock              = *LOCAL.db2dev.030815061638
  Node lock wait occurred on               = 0
  Lock object type                         = Row
  Lock mode                                = Exclusive Lock (X)
  Lock mode requested                      = Exclusive Lock (X)
  Name of tablespace holding lock          = PSAPBTABD
  Schema of table holding lock             = SAPR3
  Name of table holding lock               = TPLOGNAMES
```

```
    Lock wait start timestamp               = 08-15-2003 15:08:20.302356
    Lock is a result of escalation           = NO
  List Of Locks
    Lock Object Name                        = 29204
   Node number lock is held at = 0
   Object Type                              = Table
   Tablespace Name                          = PSAPBTABD
   Table Schema                             = SAPR3
   Table Name                               = TPLOGNAMES
   Mode                                     = IX
   Status                                   = Granted
   Lock Escalation                         = NO
  Lock Escalation                          = NO
```

2. 事件监控方式

当使用事件监控器进行锁的监控时，只能监控死锁(死锁的产生是由于锁请求冲突而不能结束事务，并且该请求冲突不能够在本事务内解决。通常是两个应用程序互相持有对方所需要的锁，在得不到自己所需要的锁的情况下，也不会释放现有的锁)的情况，具体步骤如下：

创建事件：

```
    db2 create event monitor dlock for deadlocks with details write to file
'$HOME/dir'
    db2 set event monitor dlock state 1
```

查看具体输出：

```
    db2evmon -db sample -evm dlock
    Deadlocked Connection ...
      Deadlock ID:   4
      Participant no.: 1
      Participant no. holding the lock: 2
      Appl Id: G9B58B1E.D4EA.08D387230817
      Appl Seq number: 0336
      Appl Id of connection holding the lock: G9B58B1E.D573.079237231003
      Seq. no. of connection holding the lock: 0126
      Lock wait start time: 06/08/2005 08:10:34.219490
      Lock Name       : 0x000201350000030E0000000052
      Lock Attributes : 0x00000000
      Release Flags   : 0x40000000
      Lock Count      : 1
      Hold Count      : 0
```

```
      Current Mode    : NS  - Share (and Next Key Share)
      Deadlock detection time: 06/08/2005 08:10:39.828792
      Table of lock waited on      : ORDERS
      Schema of lock waited on     : DB2INST1
      Tablespace of lock waited on : USERSPACE1
      Type of lock: Row
      Mode of lock: NS  - Share (and Next Key Share)
      Mode application requested on lock: X   - Exclusive
      Node lock occured on: 0
      Lock object name: 782
      Application Handle: 298
      Deadlocked Statement:
        Type     : Dynamic
        Operation: Execute
        Section  : 34
        Creator  : NULLID
        Package  : SYSSN300
        Cursor   : SQL CURSN300C34
        Cursor was blocking: FALSE
        Text     : UPDATE ORDERS  SET TOTALTAX = ?, TOTALSHIPPING = ?, LOCKED = ?,
TOTALTAXSHIPPING = ?, STATUS = ?, FIELD2 = ?, TIMEPLACED = ?, FIELD3 = ?, CURRENCY
= ?, SEQUENCE = ?, TOTALADJUSTMENT = ?, ORMORDER = ?, SHIPASCOMPLETE = ?,
PROVIDERORDERNUM = ?, TOTALPRODUCT = ?, DESCRIPTION = ?, MEMBER_ID = ?,
ORGENTITY_ID = ?, FIELD1 = ?, STOREENT_ID = ?, ORDCHNLTYP_ID = ?, ADDRESS_ID
= ?, LASTUPDATE = ?, COMMENTS = ?, NOTIFICATIONID = ? WHERE ORDERS_ID = ?
      List of Locks:
          Lock Name                   : 0x000201350000030E0000000052
          Lock Attributes              : 0x00000000
          Release Flags               : 0x40000000
          Lock Count                  : 2
          Hold Count                  : 0
          Lock Object Name             : 782
          Object Type                 : Row
          Tablespace Name              : USERSPACE1
          Table Schema                : DB2INST1
          Table Name                  : ORDERS
          Mode                       : X   - Exclusive
          Lock Name                   : 0x00020040000029B30000000052
          Lock Attributes              : 0x00000020
          Release Flags               : 0x40000000
          Lock Count                  : 1
          Hold Count                  : 0
          Lock Object Name             : 10675
```

```
        Object Type                    : Row
        Tablespace Name                 : USERSPACE1
        Table Schema                   : DB2INST1
        Table Name                     : BKORDITEM
        Mode                         : X   - Exclusive(略去后面信息)
```

3. db2pd 监控锁

图 10-16 展示了用于锁监视的 db2pd 选项。

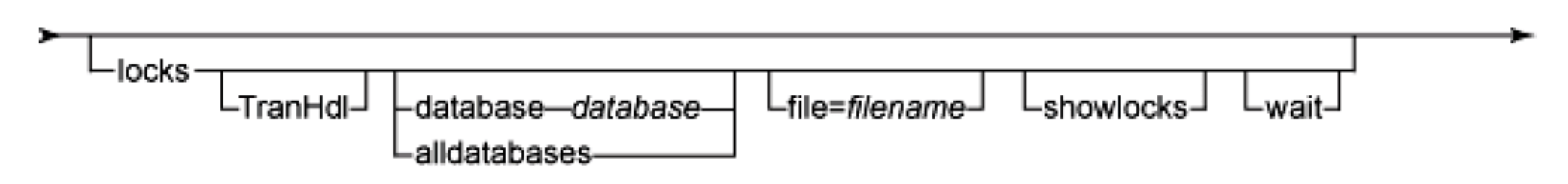


图 10-16　用于锁监视的 db2pd 选项

- TranHdl：用于指定事务句柄，以便只监视由特定事务持有的锁。
- showlocks：这个子选项将锁名称扩展成有意义的解释。对于一个行锁，该选项显示以下信息：表空间 ID、表 ID、分区 ID、页和槽。通过使用编目视图 SYSCAT.TABLES 上的一个查询，很容易将表空间 ID 和表 ID 映射到相应的表名。

下面展示了检查锁等待情形：

```
db2pd -db sample -locks wait showlocks
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:33:05
Locks:
Address    TranHdl    Lockname           Type       Mode Sts Owner      Dur
0x050A0240 6      02000600050040010000000052 Row         ..X  W   2          1
0x050A0DB0 2      02000600050040010000000052 Row         ..X  G   2          1
HoldCount  Att  ReleaseFlg
0    0x00 0x40000000   TbspaceID 2  TableID 6  PartitionID 0 Page 320 Slot 5
0    0x00 0x40000000   TbspaceID 2  TableID 6  PartitionID 0 Page 320 Slot 5
```

10.8　最大化并发性

对于好的数据库性能来说，最大化并发性非常重要。下面列出了一些详细的建议。

10.8.1　选择合适的隔离级别

选择隔离级别可以为应用程序提供可接受的最佳并发性。有几种方式可用来指定隔离级别，比如对 SQL 语句(只应用于该语句)使用 CURRENT ISOLATION 专用寄存器(应用于连接)，对 JDBC 连接对象进行指定(应用于连接)等。

10.8.2 尽量避免锁等待、锁升级和死锁

通过监控锁升级的发生(通过 DB2 状态监控器、db2diag.log、Windows 事件浏览器或其他性能监控器，确保锁列表和 MAXLOCK DB2 配置参数足够大。锁列表大小不足将会导致 DB2 尝试将大量行锁“升级”到单个表级别的锁。如果升级失败，就会导致死锁；如果升级成功，又会极大地影响到并发性。

10.8.3 设置合理的注册变量

DB2 从 V8 以后先后引入了 3 个 DB2 注册变量——DB2_EVALUNCOMMITTED、DB2_SKIPDELETED 和 DB2_SKIPINSERTED 来提高并发性。为什么要引入这 3 个变量呢？在 DB2 没有这 3 个变量前，如果一个用户正在更改(update)、插入(insert)或删除(delete)一行，那么 DB2 会在这一行加上排它锁(eXclusive)，别的用户不能读写，除非使用 UR 隔离级别。

其实目前市场上除了 Oracle 外所有的数据库包括 DB2、Informix、SQL Server 和 Sybase 对锁的控制都是这种方式。而 Oracle 由于有回滚段(rollback segment)，所以在 Oracle 数据库中对于 insert 一行，回滚段记录该插入记录的 rowid；对于 update 操作，回滚段记录更新字段的旧值(before image)；对于 delete 操作，回滚段记录整行的数据。由于 Oracle 有了回滚段，所以可以实现多版本读。所以在用 Oracle 数据库开发时，很少关注锁的情况，因为大部分情况下你都是可以读的，只不过有的时候大不了读以前的“before image”罢了。所以很多使用 Oracle 开发的用户在转向 DB2 开发时，都特别郁闷。而 DB2 为了改善应用程序并发性，从 DB2 V8 以后就陆续引入了这 3 个变量。这 3 个变量也是 DB2 的客户提出要求 IBM 去改进的，这种需求最初是 SAP 提出的。这三个变量并不会改变锁的本质，只不过是了解它们的工作方式和机制，以使我们根据我们的业务逻辑来合理地设置调整以提高应用程序的并发性。

下面我们先通过一个例子来说明没有这 3 个变量之前的一些锁的情况。假设 t1 表中有 5 条记录，分别为 11、22、33、44、55。其中第 2 条记录 22 被删除了，现在有一个 session1 要重新插入一条新的记录 22；同时第二个 session 2 执行了 db2 select * from t1 where id >11 and id<44，正常的它应该检索到 33 这条记录，但是由于现在插入的记录 22 也包含在这个谓词限定范围内，这个时候 session 2 处于 lockwait 状态。

```
Session 1                                  Session 2
db2 +c INSERT INTO t1 VALUES(22)           db2 CONNECT TO SAMPLE
db2 SELECT * FROM t1 WHERE id >11 and id <44   db2 CONNECT TO SAMPLE
```

我们通过监控看到的效果如图 10-17 所示。

图 10-17　监控结果

从 DB2 的角度来说这好像是合理的，但是从用户和业务逻辑角度来说希望这个时候能够读取到数据，那么怎么解决这个矛盾呢？下面我们来仔细讲解这 3 个变量。

DB2_EVALUNCOMMITTED

DB2 V8.1.4 版本中首次引入了 DB2_EVALUNCOMMITTED 这个 DB2 注册表变量。当它被启用(=TRUE | ON | YES | 1)时，它将修改 DB2 中只读查询的行为，使之允许在索引扫描(必须是 type-2 索引，对于 type-1 索引该特性不受支持)或表访问时推迟锁，直到限定语句的所有谓词都是已知的。引入这个新的注册表变量是为了可选地提高一些应用程序的并发性，其实质是允许读扫描推迟或避免行锁，直到适合特定查询的一个数据记录成为已知。

注意：

在 DB2 V8.1 和更高版本中，所有新索引都创建为 type-2 索引。一个例外是当您在已具有 type-1 索引的表上添加索引时，仅在这种情况下，新索引是 type-1 索引。要了解一个表存在什么类型的索引，执行 INSPECT CHECK 命令。要将 type-1 索引转换为 type-2 索引，执行 REORG INDEXES CONVERT 命令。

在 DB2 V8.1.4 之前或者 DB2 V8.1.4 之后但没有设置这个注册变量，DB2 将执行保守式的锁：在验证行是否满足查询的排除谓词之前，它将锁定每个被访问的行。不管数据行是否被提交，以及根据语句的谓词它是否被排除，对于索引扫描和表访问都执行这样的锁定操作。下面我们举一个简单的例子：

```
db2 create table t1(id int)
db2 insert into t1 values(11)
```

```
db2 commit
```

现在有两个 session 分别发出了下面的 SQL 语句：

```
Session 1
db2 CONNECT TO SAMPLE
db2 +c INSERT INTO t1 VALUES(22)
```

```
Session 2
db2 CONNECT TO SAMPLE
db2 SELECT * FROM t1 WHERE id = 11
```

我们查看 session 2 的状态，如图 10-18 所示。

第一个语句 DB2 +C INSERT INTO TABLE T1 VALUES (22)阻塞所有其他的扫描器，因为它持有行上的锁，如果第二个 session 执行 DB2 SELECT * FROM T1 将被阻塞，直到事务 1 提交或回滚。但是我们假设第二个语句是 DB2 SELECT * FROM T1 WHERE id=11。在此情况下，即使事务 2 与列 ID=22 中的任何值(还没有被提交)都没有关系，它也仍将被阻塞，处于锁等待(lockwait)状态。在 DB2 中，默认情况下将发生这一系列的事件，因为默认的隔离级别是 Cursor Stability(CS)。这种隔离级别表明，一个查询访问的任何一行在游标定位到该行时都必须被锁定。在语句 1 释放它用于更新表 T1 的锁之前，语句 2 不能包含表 T1 第一行上的锁。如果 DB2 知道值 ID=11 不是语句 2 的数据请求的一部分(换句话说，它在锁行之前计算了谓词)，就可以避免阻塞，这是合情合理的，因为语句 2 不会尝试锁定表中的第一行。

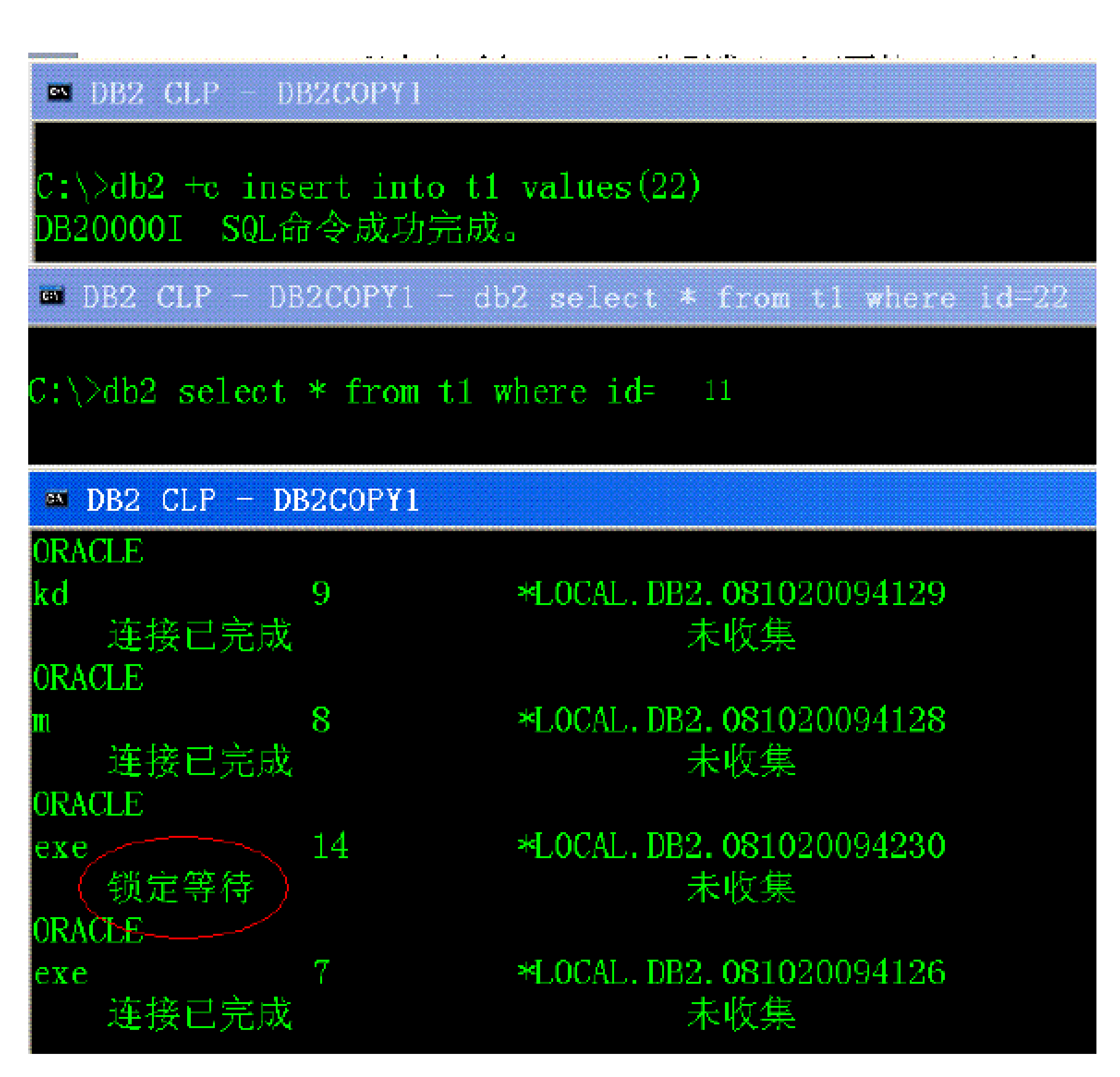


图 10-18 session 2 的状态

现在我们启用 DB2_EVALUNCOMMITTED 注册变量，该实例设置后需要重启实例。

```
db2set DB2_EVALUNCOMMITTED=ON -i
db2stop force
db2start
```

在启用该实例后，再重复刚才的实验，我们发现第二条 SQL 语句“select * from t1 where id=11”可以执行而不会被阻塞。所以 DB2_EVALUNCOMMITTED 注册变量的作用是判断该 SQL 谓词所扫描的行是否有锁，如果没有那么可以检索到数据。

当 Evaluate Uncommitted 第一次在 DB2 V8.1.4 中引入时，它带有以下限制：

- 该特性只能用于 CS 和 RS 隔离级别；
- SARGable 谓词必须存在，以便计算；
- 锁避免不适用于编目表上的扫描；
- 当扫描一个 MDC 表时，对于索引扫描，块锁可以推迟。然而，对于表扫描，块锁不会推迟；
- 被推迟的锁不会发生在正在执行在线表重组的表上；
- Index Manager 不可能在没有锁行的情况下回调 Data Manager 来取数据记录。这意味着 ISCAN-FETCH 计划不能在 Data Manager 中推迟锁(唯一的例外是对一个 MDC 表的块索引，它的 Index Evaluation 谓词是一个 ISCAN 计划)。

DB2 V8.2.2 通过去掉 DB2 V8.1.4 中第一阶段的 Evaluate Uncommitted，改进了这些缺隐陷。DB2 V8.2.2 引入了名为 DEFERISCANFETCH 的注册表变量，作为 DB2_EVALUNCOMMITTED 的新设置。启用该变量时，由该特性承担的锁避免将使用 ISCAN-FETCH 数据计划。

DB2_EVALUNCOMMITTED 变量影响 DB2 在游标稳定性(CS)和读稳定性(RS)隔离级别下的行锁机制。当你启用该功能时，DB2 可以对未提交的插入(INSERT)或者更新(UPDATE)数据进行谓词判断，如果未提交数据不符合该条语句的谓词判断条件，DB2 将不对未提交数据加锁，这样避免了因为要对未提交数据加锁而引起的锁等待状态，提高了应用程序访问的并发性。同时 DB2 会无条件进行表扫描时忽略删除的行数据(不管是否提交)。

这里分两部分来看待，“对于插入(INSERT)或者更新(UPDATE)而言，如果未提交数据不符合该条语句的谓词判断条件，那么 DB2 将不对未提交数据加锁。”，这样虽然比不上 Oracle 对于符合该条语句的谓词判断条件可以从回滚段里面读出“before image”那样做到写不阻止读，但是起码一定程度上缓解了锁的问题，不会因为插入(INSERT)或者更新(UPDATE)一条记录造成整个表都锁住。这是个进步，个人觉得也不会造成什么大的负面影响，

下面我们通过一个实验来说明这点：

```
db2 create table t1(id int)
db2 insert into t1 values(11)
db2 insert into t1 values(22)
db2 commit 现在表中有两条记录 11 和 22
```

现在两个 session 发出了下面的 SQL 语句：

```
Session 1                                Session 2
db2 CONNECT TO SAMPLE                    CONNECT TO SAMPLE
db2 +c delete from t1 where id=22        db2 SELECT * FROM t1
```

在未设置 DB2_EVALUNCOMMITTED=ON 时，session 2 肯定是处于锁等待(lockwait)状态的，现在我们设置了 DB2_EVALUNCOMMITTED=ON 后，再来看看 session 2 能否检索到数据。

```
Session 1 Session 2
                                      db2 CONNECT TO SAMPLE
                                      C:\>db2 select * from t1
db2 CONNECT TO SAMPLE                 ID
db2 +c delete from t1 where id=22     -----------
                                               11
                                      1 条记录已选择。
```

通过上面的实验，我们发现在启用 DB2_EVALUNCOMMITTED=ON 时，对于 delete 操作的处理，DB2 会无条件进行表扫描时忽略删除的行数据(不管是否提交)。个人觉得有很大的问题，通过上面的这个测试，一个事务删除一条记录并没有提交，另外一个会话查询的时候已经没有这条记录了，这相当于 UR 隔离级别。这样显然是不符合业务要求的。与其这样还不如锁住。所以用 DB2_EVALUNCOMMITTED=ON 时，对删除的时候应该注意多多测试。

现在我们在 t1 创建一个 type-2 的索引，再来做刚才的那个实验。

```
db2 create index index11 on t1(id)
db2 reorg indexes all for table oracle.t1 convert 转变为 type-2 索引
```

在两个命令行窗口(如图 10-19 所示)中分别发出下面的 SQL 语句：

```
Session 1                                                  Session 2
db2 CONNECT TO SAMPLE
db2 create index index11 on t1(id)                         db2 CONNECT TO SAMPLE
db2 reorg indexes all for table oracle.t1 convert          C:\>db2 select * from t1
db2 +c delete from t1 where id=22                          --lockwait 挂起
```

我们在另外一个窗口查看 session 2 的状态，发现 session 2 处于 lockwait 状态，如图 10-19 所示。

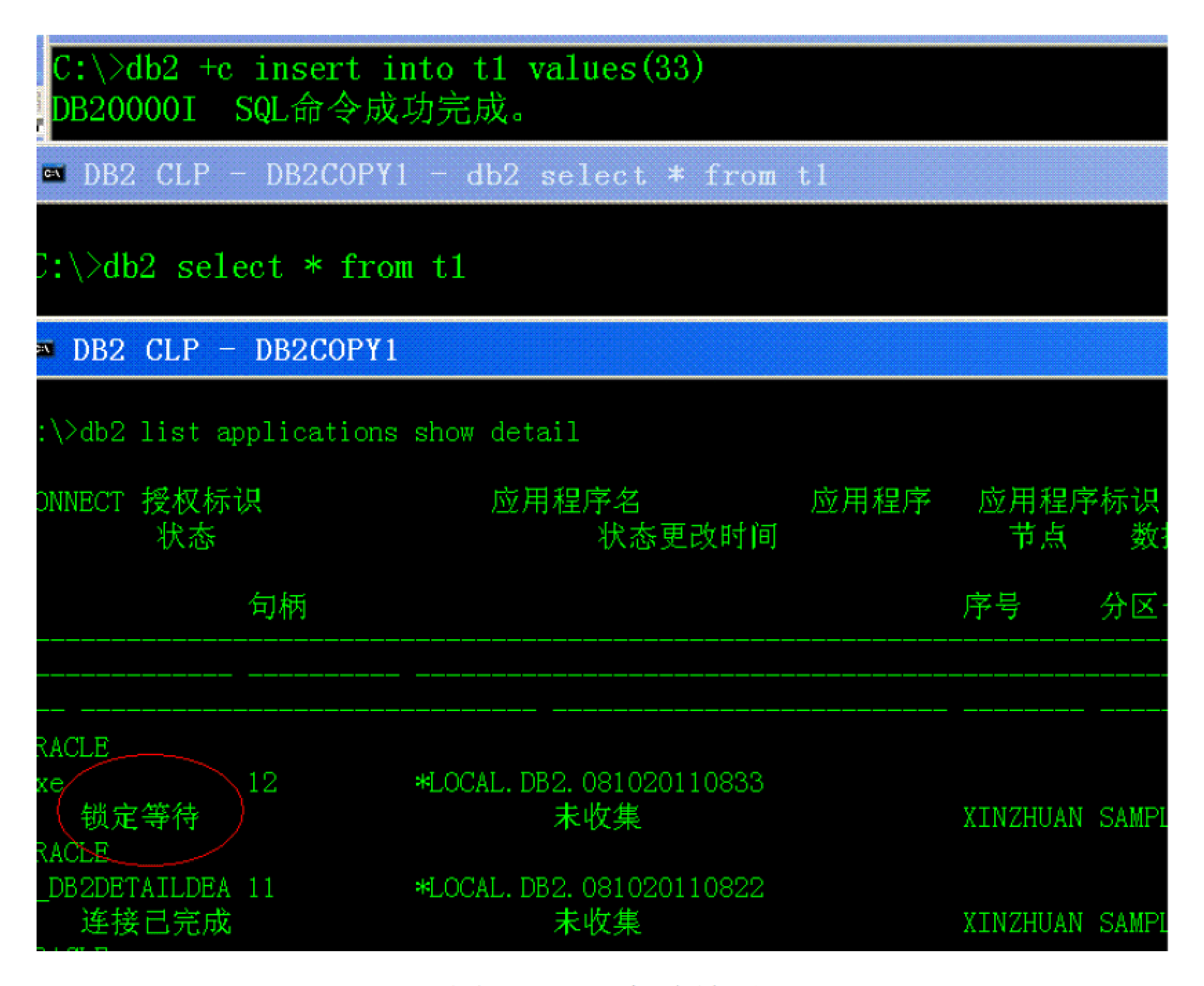


图 10-19　实验结果

当您的 DB2 环境中启用了 Evaluate Uncommitted 行为时，您应该清楚，谓词计算可能发生在未提交的数据上。而且，在表扫描访问中，被删除行被无条件忽略；而对于 type-2 索引扫描，被删除的键不会被忽略(除非您还设置了 DB2_SKIPDELETED 注册表变量，DB2_SKIPDELETED 变量我们稍后介绍)。如果您要在环境中单独设置 DB2_SKIPDELETED 注册表变量，DB2 将允许在表扫描访问时无条件地忽略被删除行，并忽略 type-2 索引扫描访问的伪删除索引键。

DB2_SKIPDELETED

变量 DB2_SKIPDELETED(被启用时=ON)，将允许使用 Cursor Stability 或 Read Stability 隔离级别的语句，在索引扫描期间无条件地跳过被删除的键，而在表访问期间则无条件地跳过被删除的行。当 DB2_EVALUNCOMMITTED 被启用时，被删除的行会被自动跳过，但是除非同时启用了 DB2_SKIPDELETED，否则 type-2 索引中未提交的伪删除键不会被跳过。

在上面的实验中，我们发现当我们设置了 DB2_EVALUNCOMMITTED 变量时，如果

表上有 type-2 索引，那么在我们读取数据时，被删除的索引键不会被忽略。这种情况下如果你希望跳过被删除的键，可以通过设置 DB2_SKIPDELETED=ON 来实现，下面我们做个实验。

```
db2set DB2 SKIPDELETED=ON -i
db2stop force
db2start
```

设置生效后，我们接着做刚才的实验。

Session 1	Session 2
	db2 CONNECT TO SAMPLE
db2 CONNECT TO SAMPLE	C:\>db2 select * from t1
db2 create index index11 on t1(id)	ID
db2 reorg indexes all for table oracle.t1 convert	-----------
db2 +c delete from t1 where id=22	11
	1 条记录已选择。

我们可以看到在设置 DB2_SKIPDELETED=ON 后，即使 t1 表上有 type-2 的索引，那么在扫描的时候仍然忽略这个删除的行。但是这个用的时候一定要结合业务逻辑使用，因为这种情况下等同于“脏读”，所以一定要多测试。

DB2_SKIPINSERTED

虽然当一个行由于一个未提交的 INSERT 而被锁的时候，这种行为是正确的。但是有些情况下应用程序的所有者希望 DB2 忽略正在等待提交的被插入的行，就好像它不存在一样(由于未提交 INSERT 的提交版本现在根本没有行，所以这是可能的)。例如，银行下午 5 点左右想统计今天的业务量，这时只是想了解大概的业务量而不是精确的，这种情况下如果启用该变量，那么，遗漏一两笔业务是可以接受的。

在 DB2 V8.2.2 中，DB2_SKIPINSERTED=OFF 是默认设置。这使得 DB2 的行为和预期的一样：扫描器一直等到 INSERT 事务提交或回滚，然后返回数据—— 这和平常一样，取决于您的应用程序以及和业务逻辑相关的数据完整性的特征，这样可能合适，也可能不合适。例如，考虑一个涉及两个应用程序的业务流程—— 例如，一个信用评级应用程序和一个信用评分引擎，这两个应用程序使用相同的一个表来交换业务信息。应用程序 A 基于一个 Web 表单将数据插入数据库，应用程序 B 读这些数据。为了加快信用审批的速度，候选者通过信用评级应用程序表单转移，信息块通过表单中的'Steps'被发送到应用程序 B(通过公共的表)。当候选者完成信用评级应用程序流程中的每个步骤时，信息被发送。在这个环境中，数据必须由第二个应用程序按照表中给出的顺序来处理，以便当接下来要读的行

要被应用程序 A 插入时，应用程序 B 必须等待，直到 INSERT 被提交。

如果设置 DB2_SKIPINSERTED=ON，DB2 将把未提交的 INSERT(只对于 CS 和 RS 隔离级别)看作它们还没有被插入。该特性提高了并发性，同时又不牺牲隔离语义。DB2 为扫描器实现了这种能力，通过锁属性和锁请求的反馈，使其忽略未提交的插入行，而不是等待。

下面我们举一个设置 DB2_SKIPINSERTED 变量前后的例子。

```
Session 1
db2 CONNECT TO SAMPLE
db2 create index index11 on t1(id)
db2 reorg indexes all for table oracle.t1 convert
db2 +c insert into t1 values(33)
```

```
Session 2
db2 CONNECT TO SAMPLE
C:\>db2 select * from t1
--挂起，处于 lockwait 状态
```

通过监控发现 session 2 处于 lockwait 状态，如图 10-20 所示。

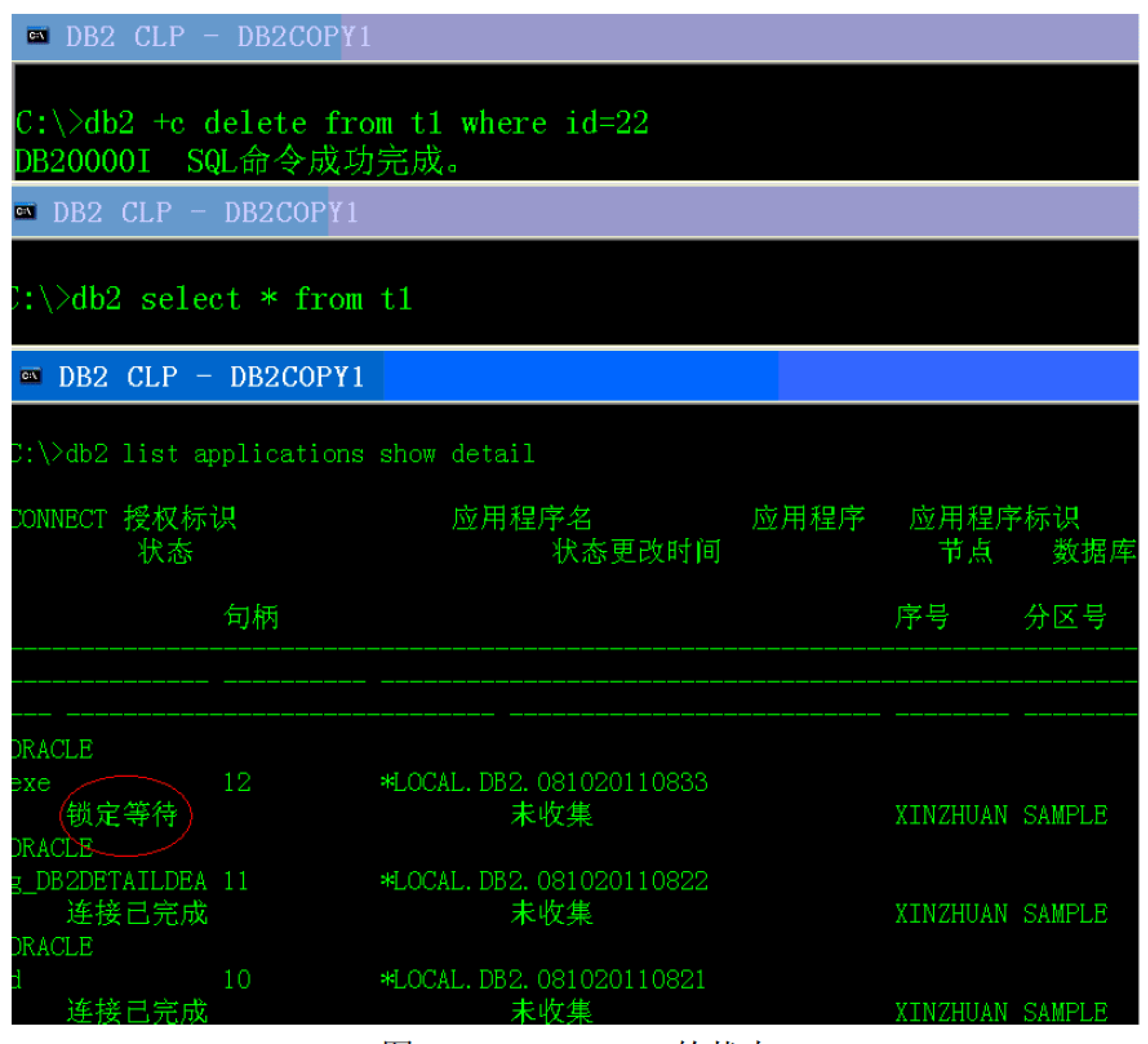


图 10-20　session 2 的状态

如果这种情况下 session2 希望能够跳过未提交 insert 的数据而得到数据，那么可以设置 DB2_SKIPINSERTED 注册变量。

```
db2set DB2_SKIPINSERTED=ON -i
db2stop force
db2start
```

然后再重复刚才的实验，我们发现这个时候，session2 可以读到数据。

```
Session 1                                          Session 2
                                                   C:\>db2 select * from t1
db2 CONNECT TO SAMPLE                              ID
db2 create index index11 on t1(id)                 -----------
db2 reorg indexes all for table oracle.t1 convert          11
db2 +c insert into t1 values(33)                           22
                                                     2 条记录已选择。
```

10.9 锁和并发总结

总的来说，这 3 个注册变量会影响到并发性。通过合理设置这些变量可以改善并发性，但是也会影响到应用程序的行为。建议在 DB2 开发设计的初期启用这些注册变量，从而在实现并发性增强后执行全面测试中的所有单元测试。

通过设置这 3 个注册变量，可以提高并发性，但是我们使用的时候一定要结合自己的业务逻辑来使用。根据个人的经验，我建议使用 DB2_EVALUNCOMMITTED=ON 和 DB2_SKIPINSERTED=ON。对于 DB2_SKIPDELETED 变量来说，使用的时候一定要充分地测试，因为它等同于使用 UR 隔离级别(注，虽然 DB2_SKIPINSERTED=ON 也等同于 UR，但是没有插入的数据反正也没有插入，读不到在业务上是可以接受的)。

总之，有了这 3 个变量，我们多了一份选择总比没有强。目前这 3 个变量都是实例级别的变量，如果能做成 SQL 级或应用级的就更好了。期待 DB2 能够在后续的版本中继续对并发作出改进。

注意：

在 DB2 V9.5 中，为了提高并发性，可以利用乐观锁定特性改善并发性，详细信息请参见《深入解析 DB2》一书。



数据库运行维护

DB2 数据库在日常运行过程中，DBA 需要经常做一些维护工作来保证数据库的正常、高效运转。这些维护工作主要包括更新统计信息、碎片整理和 rebind。统计信息是 DB2 收集的关于数据库中各个数据对象状态的信息，这些信息在收集好以后被保存在数据库系统编目表中，当应用程序或 SQL 语句对数据进行访问的时候，优化器需要根据这些统计信息来生成一个成本最低的访问方案。只有准确的统计信息才能让 DB2 数据产生最优的数据访问计划，进而进行高效的数据访问。相反地，如果数据库中只有过时的、不准确的统计信息，甚至没有相关的统计信息，那么数据的高效访问就无从谈起。所以统计信息的准确与否就显得非常重要。

随着数据被不断删除、插入和更新，数据页和索引页会变得越来越零散，数据页和索引页的物理存储顺序不再匹配其逻辑顺序，数据和索引结构的层次会变得过大，这些都会导致数据页跨越在多个页上和索引页的预读取变得效率低下。因此，需要根据数据更新的频繁程度适当地重新组织表和索引。

统计信息更新和表、索引碎片整理是 DBA 日常维护工作的一部分，在本章中我们给大家讲解如下内容：

- 统计信息更新
- RUNSTATS 更新举例
- 表、索引碎片整理
- 重新绑定应用程序包

11.1 统计信息更新

11.1.1 统计信息的重要性

DB2 优化器是 DB2 的“大脑”。而 DB2 优化器正是依靠存放在系统目录表空间中的数

据库统计信息才能做出正确的判断，估算出某个特定查询的所有访问路径的成本，随后 DB2 优化器会选择成本最低的访问路径来获取数据。您可能会问，成本究竟表示什么呢？成本即表示 DB2 优化器对执行语句所用时间的最优估计。在这里您须要注意，最常见的查询操作是 select 查询，而 DML(比如 update)或者 DDL(比如索引重建)也会执行查询操作，这同样需要通过基于成本的 DB2 优化器进行分析。

数据统计信息中的变化会影响 DB2 优化器对获取目标数据的访问路径成本的估算，从而可能选择不同的访问路径。因此，如果数据库统计信息误差过大，就有可能造成性能问题。

例如：一个有一百万行数据的表，数据库统计信息却记录这个表只有一行数据，那么 DB2根据统计信息就有可能选择全表扫描而不是索引扫描去获取数据。

下面列举了一些统计信息，这些统计信息可以帮助优化器选择最优访问策略：

- 表中的页数和非空的页数
- 表中发生行链接的数量
- 表中的行数
- 有关单个列的统计信息，比如一列中唯一值的数量
- 一个索引的聚合程度。即表中数据的存储顺序与某索引字段顺序的符合程度
- 有关索引的统计信息，比如索引级别的数量和每个索引中叶子页的数量
- 经常使用的列值的出现次数
- 列值在列中所有值中的分布状况
- 用户定义的函数(UDF)的成本估计

除以上信息外，DB2 还可以收集下列信息：索引的聚合程度、索引中叶子页数目、溢出其原始页的表行数，以及表中已填充的页数和空页数。DBA 可以参考这些信息来决定何时重组表和索引。

DB2 数据库不可能在每次数据库添加、删除、改变数据后都更新数据库统计信息，这是因为过于频繁地更新统计信息有可能造成系统性能的巨大开销；此外，如果改变的数据数量有限，那么统计信息的一些微误差有可能不会造成 DB2 重新选择新的访问路径，这就没有必要频繁更新数据库统计信息。出于上述两点原因，DB2 数据库中的统计信息并不是动态更新的，不过，DB2 提供了 RUNSTATS 命令来手工更新数据库统计信息。当对用户表中大量数据修改后，用户可以考虑在表和索引上执行 RUNSTATS 命令，用最新的信息更新系统目录表中的统计信息。

在成功执行 RUNSTATS 命令之后，静态 SQL 查询并不会使用最新的数据库统计信息，这是因为静态 SQL 的访问策略在之前执行 BIND 时就已确定，而当时使用的统计信息有可能与现在的并不一致。这时候就需要重新绑定使用静态 SQL 的应用程序，这样查询优化器

就可以根据数据库最新统计信息来选择获取数据的最佳访问策略。但是，对于使用动态SQL 的应用程序而言，则没必要进行重新绑定，因为动态 SQL 语句的访问策略是根据统计信息在运行时动态生成的。

现在，几乎所有主流数据库都使用某种方法(Oracle 数据库中调用 dbms_stats 存储过程；informix 数据库中用 update statistics；Sybase 数据库中用 update statistics)来更新系统目录统计信息，以便为其优化器提供可能的最佳信息。您可以将优化器想象成一个汽车 GPS 定位仪，它能够在系统数据所组成的莽莽深山里做行驶路径选择。目录统计信息的更新将为优化器提供最新、最准确的地图信息，以便能够获取最佳行驶路径。

如何更新统计信息

只有当进行显式请求时，对象的统计信息才会在系统目录表中被更新。有以下几种方法可以更新部分或全部统计信息：

- 使用 RUNSTATS(运行统计信息，Run Statistics)命令
- 使用带有指定的统计信息收集选项的 LOAD
- 对针对一组预先定义的系统目录视图进行操作的 SQL UPDATE 语句进行编码
- 使用“reorgchk update statistics”命令

当您不完全知道所有表名或表名实在太多、无法对每张表逐个更新策略时，进行 RUNSTATS 的最简单方法就是使用“db2 reorgchk update statistics”命令。正确的脚本如下：

```
db2 -v connect to sample
db2 -v "select tbname, nleaf, nlevels, stats time from sysibm.sysindexes"
db2 -v reorgchk update statistics on table all
db2 -v "select tbname, nleaf, nlevels, stats time from sysibm.sysindexes"
db2 -v terminate
```

我们上面所选的示例不需要表名。这一命令对所有表执行 RUNSTATS 命令。

记住：在批量数据加载后要运行 RUNSTATS 命令。

如果您知道表名并且想避免对大量表执行 RUNSTATS 命令(因为这样做可能要花很长时间)，那么一次对一张表进行 RUNSTATS 更为可取。命令如下：

```
db2 -v runstats on table  tabschema.tabname and indexes all
```

这个命令将收集该表及其所有索引(基本级别)的统计信息。

查看是否执行了 RUNSTATS 命令

要查看是否对数据库执行了 RUNSTATS 命令，一种快捷方法便是查询一些系统目录表。例如，如上面的脚本所示，可以执行下面这条命令：

```
db2 -v "select tbname, nleaf, nlevels, stats_time from sysibm.sysindexes"
```

如果还未执行过 RUNSTATS 命令，您会看到 NLEAF 和 NLEVELS 列为“－1”且 stats_time 列为“-”。如果已经执行了 RUNSTATS 命令，则这些列包含实际的数字，并且 stats_time 列将会包含时间戳记。如果您认为 stats_time 列所显示时间距离现在过久，那就需要再次执行 RUNSTATS 命令。

可以查询系统目录表中的以下列，以确定是否在表和索引上执行了 RUNSTATS 命令：

- 如果对于某个表，SYSCAT.TABLES 视图的 CARD 列显示的值为－1，或者 STATS_TIME 列显示的值为 NULL，那么表示没有对该表执行过 RUNSTATS 命令。
- 如果对于某个索引，SYSCAT.INDEXES 视图的 NLEAF、NLEVELS 和 FULLKEYCARD 列显示的值为－1，或者 STATS_TIME 列显示的值为 NULL，那么表示还没有对该索引执行过 RUNSTATS 命令。

在一个表上执行 RUNSTATS 命令时，可以有两种用户访问选项：允许读访问 ALLOW READ ACCESS 和允许写访问 ALLOW WRITE ACCESS。

在执行 RUNSTATS 命令时，如果加上 ALLOW READ ACCESS 选项，那么其他用户就只能够以只读的方式访问该表，这个选项会影响应用的并行性，因为任何想要更改表的操作这时都会处于等待状态。可以选择在使用 ALLOW READ ACCESS 选项时，同时使用 TABLESAMPLE 选项，加上这个选项只收集表的部分采样数据而不是所有数据。如果能合理选择采样数据大小，那么就可以在确保统计信息一致性的情况下，加快 RUNSTATS 的速度。

在执行 RUNSTATS 命令中时，如果加上 ALLOW WRITE ACCESS 选项，那么其他用户就可以读取或写入该表。默认情况下，RUNSTATS 命令使用的是 ALLOW WRITE ACCESS 选项。

分区数据库中的 RUNSTATS

在 DB2 V9.5 之前，执行 RUNSTATS 命令在一个数据库分区上收集统计信息，当在一个分区数据库中执行 RUNSTATS 命令，并且一个表分区位于发出 RUNSTATS 的数据库分区中时，那么 RUNSTATS 将在该数据库分区上执行。如果表分区不在该数据库分区上，那么将请求发送给数据库分区组中持有该表分区的第一个数据库分区。然后，在该数据库分区上执行 RUNSTATS 命令。

RUNSTATS 对一个分区收集统计信息，然后对所有分区所有统计信息做估算。有一个隐式的假设：每个表中的行是均匀分布在每个多分区数据库分区组中的所有分区上的。

注意：

在最新的 DB2 V9.5 版本中，在分区环境中，只需在一个分区上发出 RUNSTATS 就可以对所有分区上的数据进行统计信息更新。

11.1.2　使用 RUNSTATS 收集统计信息的原则

在下列情况下，使用 RUNSTATS 命令来收集统计信息：

- 当向表装入数据并创建了新的索引
- 当用 REORG 命令重新组织表和索引时
- 当存在大量影响表及其索引的更新、删除和插入操作时(此处的“大量”可能意味着 10%到 20%的表和索引数据都受到了影响)
- 在绑定对性能要求很好的应用程序之前
- 当预存取(prefetch size)大小发生变化时
- 当在表中创建新的索引时。如果自从上次在表中运行 RUNSTATS 以来尚未修改表，那么只需要对新的索引执行 RUNSTATS
- 当您想要比较当前和先前统计信息时。如果定期更新统计信息，那么可以及早发现性能问题
- 使用 RUNSTATS 命令来收集关于 XML 列的统计信息。如果仅使用 RUNSTATS 收集 XML 列的统计信息，将保留 LOAD 或上一次执行 RUNSTATS 命令已收集的非 XML 列的现有统计信息。如果先前已收集关于一些 XML 列的统计信息，那么在当前命令未收集关于该 XML 列的统计信息时，将删除先前收集的 XML 列的统计信息；在当前命令收集了关于该 XML 列的统计信息时，将替换先前收集的 XML 列的统计信息

要提高 RUNSTATS 性能并节省用来存储统计信息的磁盘空间，可以考虑仅指定需要收集其数据分布统计信息的列。

如果您没有足够的时间一次收集全部的统计信息，那么可以运行 RUNSTATS 来每次仅更新几个表、索引或统计信息视图的统计信息，并轮流完成该组对象。如果对选择性部分更新运行 RUNSTATS 期间，由于表上的活动而产生了不一致性，那么在查询优化期间将发出警告消息(SQL0437W，原因码 6)。例如，如果执行 RUNSTATS 来收集表分布统计信息，以及在某个表活动后，再次执行 RUNSTATS 来收集该表的索引统计信息，那么可能发生这种情况。如果由于表上的活动产生了不一致并且在查询优化期间检测到这些不一致，那么发出该警告消息。当发生这种情况时，应再次运行 RUNSTATS 来更新分布统计信息。

要确保索引统计信息和表同步，执行 RUNSTATS 来同时收集表和索引统计信息。索引统计信息保留自上次运行 RUNSTATS 以来收集的大部分表和列的统计信息。如果自上次收

集该表的统计信息以来已对该表做了大量修改，那么只收集该表的索引统计信息将使两组统计信息不能在所有节点上都同步。

在生产系统中调用 RUNSTATS 可能会对生产工作负载的性能产生负面影响。RUNSTATS 命令现在支持优先级选项，在执行较高级别的数据库活动期间，可以使用优先级选项来限制执行 RUNSTATS 的性能影响。

收集视图的统计信息时，将收集所有包含该视图引用的基本表的数据库统计信息。

可以考虑采用以下技巧来提高 RUNSTATS 的效率和已收集的统计信息的准确性：

- 仅对用来连接表的列或 WHERE、GROUP BY 以及查询的类似子句中的列收集统计信息。如果对这些列建立了索引，那么可以用 RUNSTATS 命令的 ONLY ON KEY COLUMNS 子句指定列。
- 为特定表和表中特定列定制 num_freqvalues 和 num_quantiles 的值。num_freqvalues 提供了重复最多的列和数据值的信息。num_quantiles 提供了数据值对于其他值而言是如何分布的有关信息。
- 使用 SAMPLED DETAILED 子句通过抽样计算详细的索引统计信息，这样就可以减少为获得详细索引统计信息而执行的后台计算量，使用了 SAMPLED DETAILED 子句可以减少收集统计信息所需要的时间，并在大多数情况下产生足够的精度。
- 当创建已装载数据的表的索引时，添加 COLLECT STATISTICS 子句来在创建索引时创建统计信息，这一技巧在 Oracle 环境下也同样适用。
- 当添加或删除了大量数据，或更新了收集其统计信息的列中的数据时，需要再次执行 RUNSTATS 命令来更新统计信息。
- 在 DB2 V9.5 之前，因为 RUNSTATS 仅收集单个数据库分区的统计信息，所以，如果数据不是在所有数据库分区中一致分发的，那么统计信息将不太准确。如果您怀疑存在不一致的数据分发，那么您可能想要在执行 RUNSTATS 之前使用 REDISTRIBUTE DATABASE PARTITION GROUP 命令来在各数据库分区之间重新分发数据。

您可以通过比较查询 RUNSTATS 之前和之后的 SQL 语句的 EXPLAIN 输出，来确定运行 RUNSTATS 对于访问计划的影响。

完成每一条 RUNSTATS 语句之后，您都应该执行显式的 COMMIT 命令。COMMIT 将释放锁，并避免在收集多个表的统计信息时填写日志。

在用 RUNSTATS 收集了统计信息之后，要使用 BIND 命令或 REBIND 命令重新绑定包含了静态 SQL 的应用程序包(并可以选择重新解释其语句)。db2rbind 命令可用于重新绑定数据库中的所有应用程序包。使用 FLUSH PACKAGE 命令来删除程序包缓存器(package

cache)中当前所有缓存的动态 SQL 语句(db2 flush package cache dynamic)，并强制隐式地编译下一请求。

11.1.3　减小 RUNSTATS 对系统性能影响的策略

调整 STAT_HEAP_SZ 数据库配置参数

统计信息堆(STAT_HEAP_SZ)的大小指定了使用RUNSTATS命令收集统计信息中所用内存堆的最大值。它是在启动 RUNSTATS 命令时分配的，然后当它完成时释放。STAT_HEAP_SZ 是代理私有内存的一部分。因此，在收集统计信息时，最好增大 STAT_HEAP_SZ 参数，以便能将更多的统计信息放入这个堆中。显而易见，处理较大的表也需要更多的内存。当执行包含 SAMPLED DETAILED 选项的 RUNSTATS 命令时，将会额外多占用 2MB 的 statistics 堆空间，因此必须分配更多内存，这样才能确保 RUNSTATS 命令能够更快完成。

减小 RUNSTATS 对系统性能影响的策略

- 一次仅在少数表和索引上运行 RUNSTATS，在整组表中循环运行。
- 仅指定将收集其数据分布统计信息的那些列。仅指定那些谓词中所使用的列。
- 对于不同的表，在不同的会话上执行多个并发的 RUNSTATS 命令。
- 仅在那些迫切需要提高当前工作负载性能的关键表上执行 RUNSTATS 命令。避免在不需要它的表上运行 RUNSTATS 命令。
- 根据表中数据发生改变的速度，调整 RUNSTATS 命令定期执行的频率。
- 根据 RUNSTATS 命令在该表上完成运行的速度，调整 RUNSTATS 的频率和采样数据的多少。
- 仅在系统活动量少的时候，安排执行 RUNSTATS。
- 调整(Throttling)RUNSTATS，以便最大程度地减少它对系统资源的需求。

仅在系统活动量少的时候安排执行 RUNSTATS 命令，这是最大程度地减少系统影响的一个好方法。然而，对于一个 24×7 的系统，系统中可能没有可用的时间窗口或活动量少的时候。处理该情形的一种方法就是使用 RUNSTATS 的 Throttling 选项。

Throttling 选项将根据当前的数据库活动级别，来限制 RUNSTATS 命令所占有的资源数量。DB2 中，在调整时，UTIL_IMPACT_LIM 与 UTIL_IMPACT_PRIORITY 参数的配合使用确定了 RUNSTATS 命令的行为。UTIL_IMPACT_PRIORITY 关键字被用于 RUNSTATS 命令的子句中，而 UTIL_IMPACT_LIM 则是一个实例配置参数。

UTIL_IMPACT_LIM 参数是指允许所有运行的程序对于实例的工作负载产生影响的百分比。如果 UTIL_IMPACT_LIM 是 100(默认值)，则不用调整诸如 RUNSTATS 之类的运行

程序所消耗的资源。例如，如果将 UTIL_IMPACT_LIM 设置为 10，那么正在运行的 RUNSTATS 命令就被限定消耗 10%以下的工作负载。

UTIL_IMPACT_PRIORITY 关键字可充当一个开关，它指定 RUNSTAT 命令是否使用这种调整策略。

例 11-1 UTIL_IMPACT_PRIORITY 关键字的使用。

```
RUNSTATS ON TABLE db2admin.department AND INDEXES ALL UTIL_IMPACT_PRIORITY 10
```

其中，10 是调整 RUNSTATS 的优先权或级别。

当 UTIL_IMPACT_LIM 不是 100，而且用 UTIL_IMPACT_PRIORITY 调用 RUNSTATS 时，将会对该值进行调整。如果没有为 UTIL_IMPACT_PRIORITY 指定优先权，且 util_impact_lim 不是 100，那么 RUNSTATS 将使用默认的优先权 50，并将据此进行调整。如果在 RUNSTATS 命令中省略 UTIL_IMPACT_PRIORITY 关键字，那么不会对正在运行的 RUNSTATS 命令做资源调整。如果指定了优先权，但是将 UTIL_IMPACT_LIMIT 设置为 100，那么也不会对正在运行的 RUNSTATS 命令做资源调整。将优先权设置为零，也能不对正在运行的 RUNSTATS 命令做调整。

当定义了 RUNSTATS 调整(Throttling)，并且该调整可操作时，RUNSTATS 命令通常会花费更长时间，但是对系统产生的影响会相对少一些。

11.1.4 DB2 自动统计信息收集

DB2 自动统计信息收集是在 DB2 UDB Version 8.2 中引入的。自动统计信息收集是完全自动进行表维护解决方案的一部分。其目标是允许 DB2 确定工作负载需要哪些统计信息，并定期在后台自动运行 RUNSTATS 命令，以便按需更新统计信息。

为了设置 SAMPLE 数据库自动进行统计信息收集，需要为自动维护开关设置数据库配置参数，如下所示：

```
db2 update db cfg for SAMPLE using AUTO_MAINT ON
db2 update db cfg for SAMPLE using AUTO TBL MAINT ON
db2 update db cfg for SAMPLE using AUTO_RUNSTATS ON
```

图 11-1(只显示统计信息收集和配置选项)展示了统计信息收集和配置的自动维护命令的层次结构和相关性。在该结构中，可以在最高层次快速关闭自动维护参数 AUTO_MAINT，而不丢失较低层的配置设置，比如 AUTO_RUNSTATS。

如果需要自动化统计信息配置，那么可以打开参数 AUTO_STATS_PROF 和 AUTO_PROF_UPD。通过对自动化统计信息配置来确定何时和如何更进一步收集统计信息。统计信息配置文件是自动生成的，自动统计信息收集过程将用它来调度 RUNSTATS。

可以用内部算法来比较新收集的统计信息与已保存的一组统计信息，并基于某些触发条件发出包含抽样的 RUNSTATS。

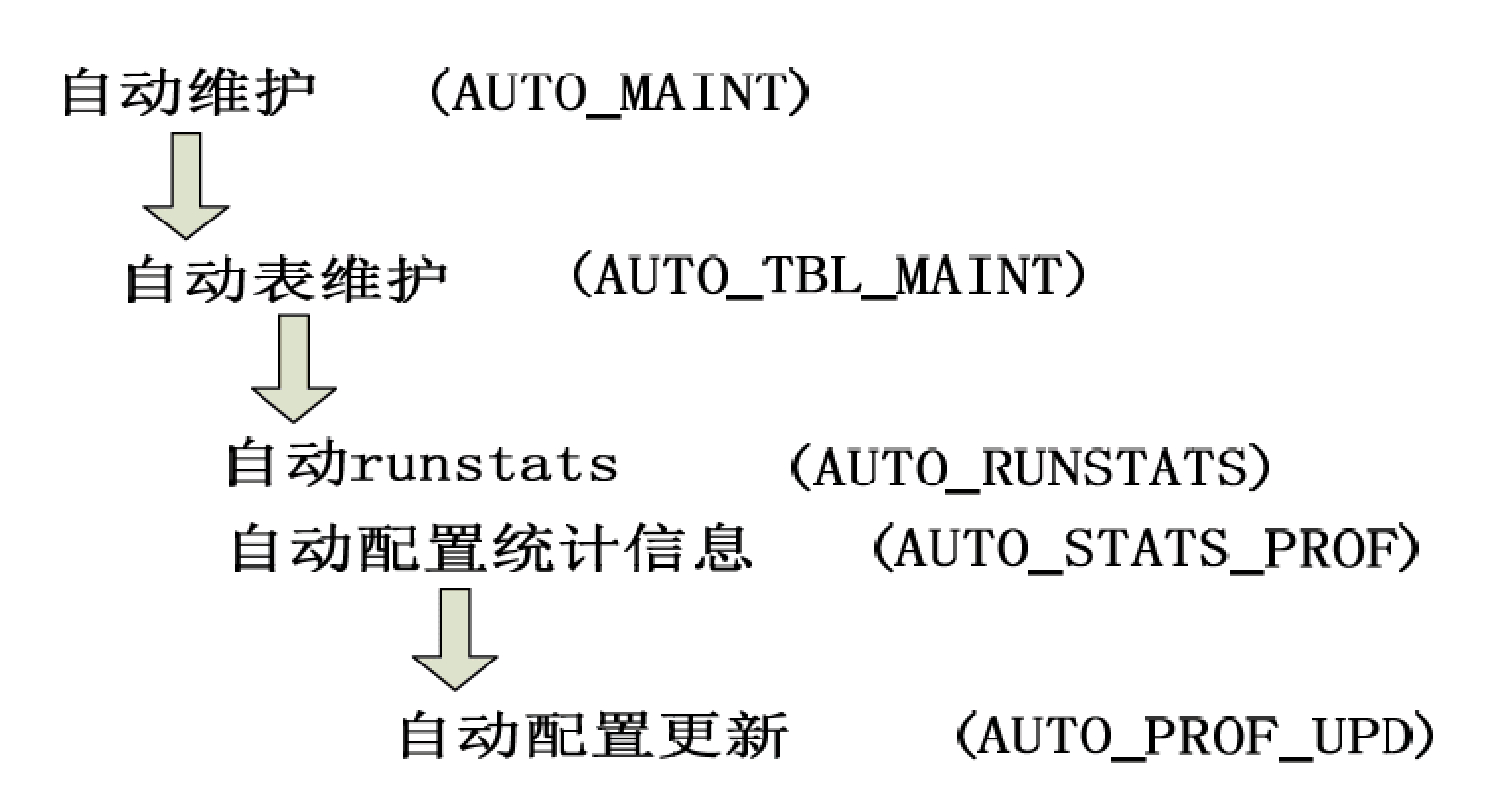


图 11-1　自动维护设置

当启用自动统计信息配置时，数据库活动的有关信息被收集并存储在查询反馈库中。然后，基于查询反馈库中的数据生成统计配置文件。

为了允许自动生成统计信息配置文件，需要设置两个数据库配置参数：

- db2 update db cfg for SAMPLE using AUTO_STATS_PROF ON 打开该参数将启动查询反馈数据的收集。
- db2 update db cfg for SAMPLE using AUTO_PROF_UPD ON 打开该参数，指定使用分析查询反馈数据的 DB2 中的建议来更新 RUNSTATS 配置文件。

在触发自动统计信息配置之前，必须通过运行 SYSINSTALLOBJECTS 存储过程创建查询反馈库。

按照下列方式调用该存储过程：

```
call SYSINSTALLOBJECTS( toolname , action , tablespacename , schemaname )
```

其中，toolname 是“ASP”或“AUTO STATS PROFILING”；对于 action，“C”表示创建，“D”表示删除。

例如，要创建反馈库，需要运行下列存储过程：

```
call SYSINSTALLOBJECTS('ASP', 'C', 'USERSPACE1', '')
```

该过程将创建下列表、存储建议以及与查询执行过程中碰到的谓词有关的信息：

- SYSTOOLS.OPT_FEEDBACK_QUERY
- SYSTOOLS.OPT_FEEDBACK_PREDICATE
- SYSTOOLS.OPT_FEEDBACK_PREDICATE_COLUMN

- SYSTOOLS.OPT_FEEDBACK_RANKING
- SYSTOOLS.OPT_FEEDBACK_RANKING_COLUMN

当禁用 AUTO_PROF_UPD 参数时，可以将建议存储在 SYSTOOLS.OPT_FEEDBACK_RANKING 表中。然后，当手动更新 RUNSTATS 配置文件时，就可以查看该表中所存储的建议。

DB2 自动统计信息更新在实际运行中一般不使用，因为它是由数据库内部控制的，我们无法操纵，它很可能在您不期望出现的时候开始运行。在实际运行中，一般我们都根据我们的业务逻辑编写如下所示的脚本，在合适的时间调度执行。

```
#!/bin/ksh
 # Source the db2 environment variables.
 . @HOME@/sqllib/db2profile
 LIST1=/tmp/tempreorg1
 LIST2=/tmp/tempreorg2
 LOG=@HOME@/schema/utilities/reorg.log
 {
 rm $LIST1 $LIST2 2>> $LOG
 DATABASE=$1
 #
 db2 -v "connect to ${DATABASE}"
 #
 # get a list of tables
 db2 "select tabname from syscat.tables where
 tabschema='@SCHEMA@' and type='T'">$LIST1
 if [ $? -ne 0 ]
 then
 print "an error occurred on connect"
 exit 1
 fi
 #
 # delete the first three lines
 # delete the line that contains "record"
 #
 sed '1,3d' $LIST1 |sed '/record(s)/d' > $LIST2
 #
 #
 print "begin reorg"
 #
 for i in `cat $LIST2`
 {
 # print reorg table $i
```

```
    db2 -v "reorg table @SCHEMA@.$i"
    if [ $? -ne 0 ]
    then
    print "An error occurred in reorg"
    exit 1
    fi
    db2 -v "runstats on table @SCHEMA@.$i with distribution and detailed indexes all"
    if [ $? -ne 0 ]
    then
    print "an error occurred in runstats"
    exit 1
    fi
    }
    print "reorg SUCCESSFUL"
    rm $LIST1 $LIST2
    } > $LOG
```

11.2　Runstats 更新举例

11.2.1　RUNSTATS 更新示例

下面举一些例子说明如何使用 RUNSTATS 收集统计信息。

注意：

在 RUNSTATS 语法中，必须使用全限定的表名 schema.table-name 和全限定的索引名 schema.index-name。您可以在所有列上，或者仅仅在某些列或列组(除了 LONG 和 LOB)上执行 RUNSTATS。如果没有指定特定列的子句，系统会使用默认的 ON ALL COLUMNS 子句。

例 11-2　收集所有列上的统计信息。

```
RUNSTATS ON TABLE db2admin.department ON ALL COLUMNS
```

这等同于：

```
RUNSTATS ON TABLE db2admin.department
```

例 11-3　收集单个列上的数据库统计信息。

```
RUNSTATS ON TABLE db2admin.department ON COLUMNS(deptno, deptname)
```

例 11-4　展示了如何收集关键列(key column)上的统计信息。短语“关键列(key column)”表示构成表上所定义索引的列。如果没有索引存在，这条命令不会收集任何列的统计信息。

例 11-5 收集关键列上的数据库统计信息。

```
RUNSTATS ON TABLE db2admin.department ON KEY COLUMNS
```

例 11-6 收集关键列上和一个非关键列上的数据库统计信息。

```
RUNSTATS ON TABLE db2admin.department ON KEY COLUMNS AND COLUMNS(deptname)
```

例 11-7 收集表和索引上的数据库统计信息，不包含分布统计信息。

```
RUNSTATS ON TABLE db2admin.department AND INDEXES ALL
```

例 11-8 收集表上的数据库统计信息以及索引上的详细统计信息，不包含分布统计信息。

```
RUNSTATS ON TABLE db2admin.department AND DETAILED INDEXES ALL
```

例 11-9 只收集 3 个指定索引上的数据库统计信息(不收集表统计信息)。

```
RUNSTATS ON TABLE db2admin.department FOR INDEXES db2admin.INX1,
db2admin.INX2, db2admin.INX3
```

例 11-10 只收集所有索引上的数据库统计信息。

```
RUNSTATS ON TABLE db2admin.department FOR INDEXES ALL
```

如果创建新的索引，而在最后一次执行 RUNSTATS 以后还未修改相应的表，那么您可以只收集索引上的数据库统计信息。在创建索引时，还可以收集数据库统计信息。

11.2.2 收集分布式统计信息

当您确定表中数据分布不均匀时，可以运行包含 WITH DISTRIBUTION 子句的 RUNSTATS 命令。系统目录表中的统计信息通常包含关于表中最高和最低值的信息，而优化器假定数据值是在两个端点值之间均匀分布的。然而，如果数据值彼此之间差异较大，**或者群集在某些点上，**或者是碰到许多重复的数据值，那么优化器就无法选择一个最佳的访问路径，除非收集了分布统计信息。使用 WITH DISTRIBUTION 子句还可以帮助查询处理没有参数标记(parameter marker)或主机变量的**谓词**，因为优化器仍然不知道运行时的值是有许多行，还是只有少数行。

下面的例子说明了使用 RUNSTATS 来收集包含数据分布信息的系统目录表统计信息的不同方法。

例 11-11 收集表和索引上的数据库统计信息，包含分布统计信息。

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL
```

例 11-12　收集表上的数据库统计信息以及索引上的详细统计信息，包含分布统计信息。

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND DETAILED INDEXES ALL
```

例 11-13　收集选定列中包含分布的数据库统计信息。

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS(deptno,
deptname)
```

例 11-14　只收集表上的数据库统计信息，包含 deptno 和 deptname 上的基本列统计信息以及 mgrno 和 admrdept 上的分布统计信息。

```
RUNSTATS ON TABLE db2admin.department ON COLUMNS(deptno, deptname)
  WITH DISTRIBUTION ON COLUMNS(mgrno, admrdept)
```

例 11-15　收集构成索引的所有列以及两个非索引列中包含分布的数据库统计信息。

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON KEY
  COLUMNS AND COLUMNS(admrdept, location)
```

11.2.3　包含频率和分位数统计信息的 RUNSTATS

在执行包含 WITH DISTRIBUTION 子句的 RUNSTATS 时，会根据 RUNSTATS 命令中给定的选项选择一组频率(frequency)和分位数(quantile)的统计信息。

RUNSTATS 收集两种类型的数据分布统计信息：频率统计信息和分位数统计信息。

频率统计信息的默认值由 *num_freqvalues* 数据库配置参数控制，该值提供了重复最多的列和数据值的信息。其默认值是 10，建议将这个值设置在 10 到 100 之间。如果将 *num_freqvalues* 设置为零，则不保留任何频率值的统计信息。

分位数统计信息的默认值由 *num_quantiles* 数据库配置参数控制，该值提供了数据值对于其他值而言是如何分布的有关信息。*num_quantiles* 数据库配置参数指定应将列数据值分成的组数。其默认值是 20，建议将该值设置在 20 到 50 之间。如果将这个参数设置为零或“1”，则不收集任何分位数统计信息。

如果没有在 RUNSTATS 命令的列或表级别上指定 num_freqvalues 和 num_quantiles，那么 num_freqvalues 的值将从 num_freqvalues 数据库配置参数中获取，而 num_quantiles 的值将从 num_quantiles 数据库配置参数中获取。

可以为单个列或一组列修改频率和分位数统计信息的精确度。提高分布统计信息的精确度将导致更大的 CPU 和内存消耗，并占用更多的系统目录表空间。对于这些分布统计信息，只考虑对拥有选择谓词的最重要的查询而言最为重要的列。

当出现下列任何一种条件时，RUNSTATS 将不收集分布统计信息：

- 当将 *num_freqvalues* 配置参数设置为零(0)，以及将 *num_quantiles* 数据库配置参数设置为零(0)或 1 时。
- 当每个数据值是唯一的时候。
- 当该列是 LONG、LOB 或结构化列时。
- 如果列中只有一个非空值。
- 声明的临时表。

下列例子说明了使用 RUNSTATS 来收集目录统计信息和指定 *num_freqvalues* 和 *num_quantiles* 的不同方法：

假设 *num_freqvalues* 数据库配置参数的值是 10，*num_quantiles* 数据库配置参数的值是 20。

例 11-16 收集包含分布统计信息的数据库统计信息。

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL
```

将 num_freqvalues 参数设置为 10，num_quantiles 参数设置为 20，因为在命令行上没有指定 num_freqvalues 和 num_quantiles 参数。

例 11-17 仅收集表上的数据库统计信息，其中使用指定的 num_freqvalues 以及从数据库配置设置选择 num_quantiles 收集所有列上的分布统计信息。

```
 RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION DEFAULT
   NUM_FREQVALUES 40
```

将 num_freqvalues 参数设置为 40，num_quantiles 参数设置为 20。

例 11-18 收集表上的数据库统计信息，包含列 deptno 和 deptname 上的分布统计信息。单独为 deptname 列设置分布统计信息的范围，而 deptno 列使用公共的默认值。并且还为两个索引 IDX1 和 IDX2 收集数据库统计信息。

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS
   (deptno, deptname NUM_FREQVALUES 50 NUM_QUANTILES 100) DEFAULT
   NUM_FREQVALUES 5 NUM_QUANTILES 10 AND INDEXES db2admin.IDX1,
db2admin.IDX2
```

对于 deptname 列，num_freqvalues 是 50，num_quantiles 是 100。

对于 deptno 列，num_freqvalues 是 5，num_quantiles 是 10。

例 11-19　收集所有索引上的数据库统计信息，包含列 deptname 上的分布统计信息。未列出的列上的分布统计信息将被清除。

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS
  (deptname NUM_FREQVALUES 20 NUM_QUANTILES 40)
   AND INDEXES ALL
```

对于 deptname 列，num_freqvalues 是 20，num_quantiles 是 40。

例 11-20　收集所有索引以及列 deptno 和 deptname 上的数据库统计信息。deptno 列的 num_freqvalues 和 num_quantiles 值将从默认值中获得。

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS
 (deptno, deptname NUM_FREQVALUES 20 NUM_QUANTILES 40) DEFAULT
   NUM_FREQVALUES 0 NUM_QUANTILES 0  AND INDEXES ALL
```

对于 deptname 列，num_freqvalues 是 20，num_quantiles 是 40。

对于 deptno 列，num_freqvalues 是 0，num_quantiles 是 0。

其他所有列不包含任何统计信息。

11.2.4　包含列组统计信息的 RUNSTATS

列组(Column Group)统计信息将获得一组列的不同值组合的数目。通常，DB2 优化器可用的基本统计信息不检测数据相关性。列组的使用将给多个谓词的联合选择提供更准确的估计。列组统计信息假设数据是均匀分布的。但还无法获得列组上的分布统计信息。

与列组的基数相比，单个列的基数(cardinality)的乘积将获得更好的相关性估计。

下列例子说明了如何使用 RUNSTATS 收集捕获了列组信息的数据库统计信息。

例 11-21　收集捕获了列组的数据库统计信息。

```
RUNSTATS ON TABLE db2admin.department ON COLUMNS((deptno, deptname),
   deptname, mrgno,(admrdept, location))
```

本例中，总共有两个列组：(deptno，deptname)和(admrdept，location)。

11.2.5　包含 LIKE STATISTICS 的 RUNSTATS

当在 RUNSTATS 中指定 LIKE STATISTICS 子句时，将收集附加的列统计信息。这些统计信息存储在 SYSIBM.SYSCOLUMNS 表里的 SUB_COUNT 和 SUB_DELIM_LENGTH 列中。它们仅针对字符串列进行收集，查询优化器用它们来提高“column LIKE '%abc'”和“column LIKE '%abc%'”类型谓词的选择性估计。

下列例子说明了如何使用 RUNSTATS 收集包含 LIKE 统计信息的数据库统计信息。

例 11-22 收集所有列上的数据库统计信息并指定 VARCHAR 列上的 LIKE 统计信息。

```
    RUNSTATS ON TABLE db2admin.department ON ALL COLUMNS and COLUMNS(deptname
LIKE STATISTICS)
```

11.2.6 包含统计信息配置文件的 RUNSTATS

现在，可以在 DB2 V8.2 中为 RUNSTATS 建立一个统计信息的配置文件。统计信息配置文件是指一组选项，它预先定义了特定表上将要收集的统计信息。

当将命令参数“SET PROFILE”添加到 RUNSTATS 命令时，将在表描述符和系统目录中注册或存储统计信息配置文件。若要更新该统计信息配置文件，则可以使用命令参数“UPDATE PROFILE”。DB2 中没有删除配置文件的选项。

下列例子展示了如何使用这项功能。

例 11-23 只注册一个统计信息配置文件，不收集数据库统计信息。

```
    RUNSTATS ON TABLE db2admin.department AND INDEXES ALL SET PROFILE ONLY
```

RUNSTATS 中的子句“SET PROFILE ONLY”不收集统计信息。

例 11-24 注册一个统计信息配置文件，并执行所存储统计信息配置文件的 RUNSTATS 命令选项来收集目录统计信息。

```
    RUNSTATS ON TABLE db2admin.department AND INDEXES ALL SET PROFILE
```

例 11-25 仅修改现有的统计信息配置文件，不收集任何数据库统计信息。

```
    RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL
UPDATE PROFILE ONLY
```

例 11-26 修改现有的统计信息配置文件，并执行已更新的统计信息配置文件的 RUNSTATS 命令选项来收集数据库统计信息。

```
    RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL
UPDATE PROFILE
```

例 11-27 根据前面已注册的统计信息配置文件来查询 RUNSTATS 选项。

```
    SELECT STATISTICS_PROFILE FROM SYSIBM.SYSTABLES WHERE NAME = 'DEPARTMENT'
AND CREATOR = 'DB2ADMIN'
```

例 11-28　使用前面已注册的统计信息配置文件收集数据库统计信息。

```
RUNSTATS ON TABLE db2admin.department USE PROFILE
```

11.2.7　带有抽样的 RUNSTATS

随着数据库不断地快速增长，由于时间窗口、内存和 CPU 等约束的限制，通过全表扫描来收集数据库统计信息将会变得越来越困难。这时候可以考虑通过数据抽样，即只扫描表的一个数据子集来收集数据库统计信息。

如果一个查询试图预计总的趋势和模式，且某一误差域(margin of error)内的近似答案足以监测这些趋势和模式，那么数据抽样或许是比全表扫描更好的选择。

在 DB2 V8.1 中，引入了 SAMPLED DETAILED 子句，允许通过抽样计算详细的索引统计信息。该子句的使用将减少为获得详细索引统计信息而执行的后台计算量和所需的时间，但在大多数情况下，都会使数据足够的精确。

以下是一些关于该子句的使用的例子。

例 11-29　收集两个索引上的详细数据库统计信息，但对每个索引条目使用抽样来代替执行详细的计算。

```
RUNSTATS ON TABLE db2admin.department AND SAMPLED DETAILED INDEXES ALL
```

例 11-30　收集索引上的详细抽样统计信息和表的分布统计信息。

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON KEY
  COLUMNS AND SAMPLED DETAILED INDEXES ALL
```

在 DB2 V8.2 中，提供了对表数据进行抽样的两种新方法：行级的贝努里(Bernoulli)抽样和系统页级的抽样。

11.2.8　带有系统页级抽样的 RUNSTATS

系统页级抽样与行级抽样类似，只不过抽样的对象是页面而不是行。以 P/100 的概率选择每一页，而以 1 - P/100 的概率拒绝页的选择。在所选中的每一页中，要选择所有的行。系统页级抽样优于全表扫描或贝努里(Bernoulli)抽样的地方是它节省了 I/O。

抽样页也是预取的，所以该方法将比行级贝努里(Bernoulli)抽样更快。与不进行抽样相比，页级抽样极大地提高了性能。

RUNSTATS REPEATABLE 子句允许通过 RUNSTATS 语句生成相同的样本，只要表数据没有发生更改。为了指定该选项，用户还必须提供一个整数，以表示将用于生成样本的种子(seed)。通过使用相同的种子，可以生成相同的样本。

总之，统计信息的准确性取决于抽样率、数据倾斜(data skew)，以及用于数据抽样的数据群集。

下面是一些使用贝努里(Bernoulli)行级和系统页级抽样的 RUNSTATS 例子。

例 11-31 收集统计信息，包含 10%行上的分布统计信息。

```
    RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION TABLESAMPLE
BERNOULLI(10)
```

例 11-32 为了控制收集统计信息的样本集，以及可以重复使用相同的样本集，需要使用下列语句。

```
    RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION TABLESAMPLE
      BERNOULLI(10) REPEATABLE(1024)
```

例 11-33 收集 10%的数据页上的索引统计信息和表统计信息。请注意，只对表数据页进行抽样，而不是索引页。本例中，10%的表数据页用于表统计信息的收集，而对于索引统计信息，将使用所有的索引页。

```
    RUNSTATS ON TABLE db2admin.department AND INDEXES ALL TABLESAMPLE SYSTEM(10)
```

11.2.9 收集统计信息的其他可供选择的方法

对于数据库中的所有表，收集统计信息的一个可供选择的方法就是发出一条 REORGCHK 命令，如下所示。

例 11-34 使用 REORGCHK 收集所有表的数据库统计信息。

```
    REORGCHK UPDATE STATISTICS ON TABLE ALL
```

REORGCHK 命令的 UPDATE STATISTICS 选项的作用类似于调用 RUNSTATS 例程来更新数据库中所有表的数据库统计信息，但是所有列上的统计信息只能通过默认的 RUNSTATS 选项来收集。通过使用该命令，还可以收集单个表上或同一模式下一组表的统计信息。

例 11-35 使用 REORGCHK 收集一个表的数据库统计信息。

```
    REORGCHK UPDATE STATISTICS ON TABLE db2admin.department
```

例 11-36 使用 REORGCHK 收集一个模式的数据库统计信息。

```
    REORGCHK UPDATE STATISTICS ON SCHEMA systools
```

虽然该方法看上去是一种在多个表上收集数据库统计信息的快速方法，但是仅仅当 REORGCHK 可以在合理的时间内完成执行时，该方法才是可取的。

在表的 LOAD REPLACE 期间以及索引的创建期间，也可以进行数据库统计信息的收集。

在 DB2 UDB V8.2 中，为了在 LOAD 期间收集数据库统计信息，必须将选项“STATISTICS USE PROFILE”添加到 LOAD 控制语句中。选项“STATISTICS YES”仍然有用，但它现在已是过时的语法。若在表的 LOAD 之后在同一表上执行 RUNSTATS，那么在 LOAD 期间发出“STATISTICS USE PROFILE”的目的就是为了减少正常运行过程中占用的时间。在执行 LOAD 之前，就必须创建统计信息配置文件，它现在允许指定与 RUNSTATS 命令中相同的统计信息选项。

例 11-37　使用 LOAD 命令和 STATISTICS USE PROFILE 子句收集数据库统计信息。

```
    LOAD FROM inputfile.del OF DEL REPLACE INTO db2admin.department STATISTICS
USE PROFILE
```

大多数情况下，RUNSTATS 是在创建索引之后执行的。然而，在执行索引创建操作时，也能收集索引的统计信息。这将避免为了收集统计信息而进行的另一项索引扫描。下列例子说明了完成这项工作可以使用的一些选项：

例 11-38　收集基本的索引数据库统计信息。

```
    CREATE INDEX db2admin.inx1 ON db2admin.department(deptno) COLLECT
STATISTICS
```

例 11-39　收集扩展的索引数据库统计信息。

```
    CREATE INDEX db2admin.inx2 ON db2admin.department(deptname) COLLECT
DETAILED STATISTICS
```

例 11-40　收集扩展的索引数据库统计信息，指定使用抽样。

```
    CREATE INDEX db2admin.inx3 ON db2admin.department(deptname) COLLECT SAMPLED
DETAILED STATISTICS
```

11.2.10　RUNSTATS 总结

理解并正确使用 RUNSTATS 是维护高性能数据库的关键之一。本节讨论了 RUNSTATS 的重要性，并解释了许多使 RUNSTATS 更有用的可用选项。其中一些新选项，通过允许建立指定需要为每个表收集哪些统计信息的配置文件，简化了管理。本节所展示的示例将使您易于使用 RUNSTATS 获得最佳的数据库性能。

11.3 表和索引碎片整理

随着数据被不断删除、插入和更新，数据页和索引页会变得越来越零散，数据页和索引页的物理存储顺序不再匹配其逻辑顺序，数据和索引结构的层次会变得过大，这些都会导致数据页跨越在多个页上和索引页的预读取变得效率低下。因此，根据数据更新的频繁程度需要适当地重新组织表和索引。

11.3.1 表重组(REORG)

对表数据进行大量更改之后，逻辑上连续数据可能分散存放在很多个不连续的物理数据页中，因此数据库管理器必须执行更多的读操作来访问数据。另外，在某个表删除大量行后，由于表的高水位标记并不会发生变化，因此对该表做全表扫描时就会做很多不必要的 I/O 操作。在这样的情况下，您可以考虑重组表以回收浪费的空间和对数据进行重组。此时，既可重组系统目录表，也可以重组数据库表。

注意：

由于重组表的时间通常要比运行统计信息的时间长，因此您可以执行 RUNSTATS 以更新当前的数据统计信息，并重新绑定应用程序。如果更新的统计信息并没有改善性能，那么重组可能会有所帮助。

1. 重组表的方法

在 DB2 V8 之前要进行重组，只能停止应用，断开连接，离线重组，这对业务的高可用性会造成影响。DB2 V8 后有两种不同的表重组方法：脱机 **REORG** 和联机 **REORG**。

REORG 命令的 INPLACE 选项指定联机重组。如果未指定此选项，那么将运行脱机 REORG。

这两种重组方法各有一些优点和缺点。下面概述了这些优缺点。选择重组方法时，应考虑哪种方法提供的优点是您优先要解决的方面，以及业务是否允许。例如，如果发生故障时进行恢复比快速完成重组更重要，那么最好使用联机重组方法。

脱机重组的优点：

- 表重组速度最快，尤其在不需要重组 LOB/LONG 数据时。
- 完成时精确地维护了表和索引的集群。
- 重组表之后立即重建索引。不需要单独的步骤来重建索引。

- 可以在临时表空间中构建影子副本。这减少了包含目标表或索引的表空间中需要的空间大小。
- 允许指定并使用集群索引外的索引来重新维护数据的集群，而联机重组在存在集群索引时必须使用现有集群索引。

脱机重组的缺点：

- 表访问受限制。只有在重组的排序和构建阶段才允许应用程序对表进行只读访问。
- 由于使用影子副本方法，所以需要较大空间。
- 与联机 REORG 相比，对 REORG 过程的控制较少。不能暂停然后重新启动脱机重组。

联机重组的优点：

- 允许应用程序在 REORG 期间对表进行完全访问。
- 对 REORG 过程具有更多控制：该过程在后台异步运行，并且可以使其暂停、继续以及停止。例如，如果正在对表进行大量更新或删除操作，那么可以暂停 REORG 过程。
- 在发生故障时可恢复重组过程。
- 由于采用递增方式处理表，所以需要较少空间。

联机重组的缺点：

- 可能产生非最佳数据集群或非最佳索引集群，这取决于 REORG 期间访问表的事务类型。
- 在重组开始时重组的页可能更新次数更多，从而比重组过程中稍后重组的表具有更多碎片。
- 速度比脱机重组要慢。对于正常集群 REORG(不仅仅是空间回收)，执行联机重组的时间可能是脱机重组所用时间的 10 到 20 倍。对于同时正在对其运行应用程序的表，或者对于定义了大量索引的表，此时间可能还要长很多。
- 联机重组是一个可恢复的过程，但这会导致日志记录要求提高。可能需要极大量的日志空间，最大可为表大小的若干倍。这取决于重组期间移动的行数、对表定义的索引数以及索引的大小。
- 将维护索引而不是进行重建，因此以后可能需要重组索引。

表 11-1 对联机重组和脱机重组进行比较。

表 11-1　联机重组与脱机重组的比较

特征	脱机重组	联机重组
性能	快	慢
完成时数据的集群因子	良好	非最佳集群
并行性(对表的访问)	从不能访问到只读访问	从只读访问到完全访问
数据存储空间要求	非常大	不是非常大
日志记录存储空间要求	不是非常大	非常大
用户控制(暂停和重新启动重组过程的能力)	较少控制	较多控制
可恢复性	完全可恢复或完全不可恢复：成功或失败。	可恢复
索引重建	进行	不进行
支持所有类型的表	是	否
指定除集群索引外的索引	是	否
使用临时表空间	是	否

2. 监视表重组的进度

有关表重组当前进度的信息将写入数据库活动的历史记录文件中。历史记录文件中包含每个重组事件的记录。要查看此文件，请执行 db2 list history 命令以打开包含所重组表的数据库。

此外，也可使用表快照来监视表重组的进度。不管如何设置“数据库监视器表”开关，系统均会记录表重组监视数据。下面的示例显示了重组期间调用快照监控的输出。

```
C:\pp>db2 get snapshot for tables on testdb
                表快照
第一个数据库连接时间戳记          = 2008-10-27 00:43:16.630753
快照时间戳记                      = 2008-10-27 00:44:16.475355
数据库名称                                  = BANK
数据库路径                            = C:\DB2\NODE0000\SQL00002\
输入数据库别名                        = TESTDB
表列表
 表模式                 = ORACLE
```

```
    表名                    = ACCOUNT
    表类型                  = 用户
   数据对象页    = 1820
    读取的行              = 210000
    写入的行              = 0
    溢出                    = 0
    重组的页          = 0
    表重组信息:
      重组类型          =
             回收
             表重组
             允许读访问
             仅重组数据
      重组索引          = 0
      重组表空间        = 2
      开始时间                     = 2008-10-27 00:44:06.878339
      重组阶段          = 2 - 替换
      最大阶段              = 2
      阶段开始时间          = 2008-10-27 00:44:07.576298
      状态                  = 已完成
      当前计数器            = 0
      最大计数器            = 0
      完成                  = 0
      结束时间              = 2008-10-27 00:44:07.594767
```

3. 以脱机方式重组表

以脱机方式重组表是整理表碎片最快的方法。重组可减少表所需的空间量并提高数据访问和查询性能。

标识需要重组的表之后，可以对这些表以及表上的索引执行 REORG 命令。

要使用 CLP 重组表，请执行 REORG TABLE 命令：

```
db2 reorg table test.employee
```

要使用临时表空间 mytemp 重组表，请输入：

```
db2 reorg table test.employee use mytemp
```

要重组表并根据索引 myindex 对行进行重新排序，请输入：

```
db2 reorg table test.employee index myindex
```

要使用 SQL 调用语句重组表，请使用 ADMIN_CMD 过程发出 REORG TABLE 命令：

```
call sysproc.admin_cmd('reorg table employee index myindex')
```

在重组表之后，应收集有关表的统计信息，以便优化器具有最准确的数据来评估查询访问方案。

4. 脱机表重组的恢复

在重组替换阶段开始之前，脱机表重组是一个完全成功或完全失败的过程。这表示如果系统在排序或构建阶段崩溃，那么重组表操作将回滚，并且不会在崩溃恢复时重新进行该操作。而是必须在恢复后重新发出 REORG TABLE 命令。

如果系统在进入替换阶段后崩溃，那么重组表操作必须完成。这是因为已完成所有重组表工作，原始表可能不再可用。在崩溃恢复期间，需要已重组对象的临时文件，而不是用于排序的临时表空间。恢复操作将完全重新开始替换阶段，并且需要恢复副本对象中的所有数据。在这种情况下，SMS 表空间与 DMS 表空间之间有一个差别：必须将 SMS 对象从一个对象复制到另一个对象；而在 DMS 中，如果重组在相同表空间中进行，那么仅指向刚刚重组的对象并废弃原始表。

索引重建阶段出现崩溃表示我们已经拥有新对象，因此不重新执行任何操作。如上所述，将不再重建索引，但在崩溃恢复期间会将索引标记为无效。数据库将根据一般规则确定重建索引的时间：在数据库重新启动时或第一次访问索引时(注：这个可以通过设置 DBM 配置参数 INDEXREC 来制定索引重新创建和构建的时间，默认是 RESTART)。

5. 提高脱机表重组的性能

脱机表重组的性能在很大程度上由数据库环境的特征决定。

事实上，以 NO ACCESS 方式运行的重组表操作与以 ALLOW READ ACCESS 方式运行的重组表操作在性能方面没有任何差别。唯一的区别在于，对于 READ ACCESS 方式，DB2 在替换表之前升级对该表的锁定，因此实用程序可能必须等到现有扫描完成并释放其锁定后才能获得相应的锁。在这两种方式下，表在重组表操作的索引重建阶段不可用。

6. 提高重组性能的技巧

如果表空间中有足够的空间，那么对原始表和表的已重组副本使用相同的表空间，而不使用临时表空间。这将节省从临时表空间中复制表所用的时间。

- 考虑在重组表之前删除不必要的索引，以便在重组表操作期间维护较少的索引。
- 确保正确设置了已重组的表所在的表空间的预取大小。
- 启用 INTRA_PARALLEL，以便使用并行处理完成索引重建。

- 调整 sortheap 和 sheapthres 数据库配置参数以控制排序空间。因为每个处理器都将执行私有排序，所以 sheapthres 的值至少应为 sortheap×使用的处理器数。
- 通过调整页清除程序的数目确保尽快从缓冲池中清除脏索引页。

7. 联机表重组

联机表重组允许用户重组表的同时对该表进行完全访问。虽然联机表重组允许用户仍能对数据继续进行访问，但其重组速度比脱机表重组要慢。

进行联机表重组时，不是立即重组整个表，而是按顺序重组表的各个部分。不会将数据复制到临时表空间：在现有表对象中移动行以重新建立集群、回收可用空间并消除溢出行。

联机表重组包括以下 4 个主要阶段：

(1) 选择 N 页(SELECT N pages)。在此阶段中，DB2 将选择 N 页，其中 N 是包含要进行重组表处理的扩展数据块(extent)大小(最少 32 个连续页的空间)。

(2) 释放空间(vacate the range)。选定 N 页后，联机表重组将第一阶段选定范围内的所有行移至表中的可用页。每个被移动的行都保留一条 RP(REORG TABLE Pointer)记录，该记录包含行的新位置的 RID。将行作为包含数据的 RO(REORG TABLE Overflow)记录插入到表的可用页中。

重组表完成移动一组行后，它将等待表中进行的所有现有数据访问(例如，通过当前执行的应用程序)完成。这些现有访问称为旧扫描程序，它们在访问表数据时使用旧 RID。在此等待过程中启动的任何访问都称为新扫描程序，它们使用新的 RID 来访问数据。在所有旧扫描程序都完成后，重组表操作将清除已移动的行、删除 RP 记录并将 RO 记录转换为正常记录。

(3) 填充范围(Fill the range)。所有行的空间都空出后，将采用已重组的格式、根据使用的任何索引进行排序后的顺序并遵循定义的任何预取限制写回这些行。填充范围中的所有页后，将在表中选择下一个 N 有序页，并且循环开始该过程。

(4) 截断表(truncate table)。重组表中的所有页后，默认情况下将移动表的高水位标记以回收空间。如果指定了 NOTRUNCATE 选项，那么则不会移动表的高水位标记。

8. 联机表重组期间创建的文件

联机表重组期间，将为每个数据库分区创建一个.OLR 状态文件。此文件是名为 xxxxyyyy.OLR 的二进制文件，其中 xxxx 是池标识，而 yyyy 是十六进制格式的对象标识。此文件包含从暂停状态继续联机重组所需的信息。

状态文件包括下列信息：

- 重组类型
- 所重组的表的生存 LSN
- 要腾出的下一个范围
- 重组是为了维护数据的集群还是仅回收空间
- 用于维护数据集群的索引的标识

校验和保存在.OLR 文件中。如果该文件已损坏并导致校验和错误，或者表 LSN 与生存 LSN 不匹配，那么必须启动一个新的重组，并且将创建新的状态文件。

如果删除了.OLR 状态文件，那么重组表过程就不能继续，并且会返回 SQL2219 错误。必须启动新的重组表过程。

不应从系统中手动删除与重组过程关联的文件。

9. 恢复失败的联机表重组

通常，磁盘已满或日志记录错误之类的处理错误会导致联机表重组失败。如果联机表重组失败，那么 SQLCA 消息将写入历史记录文件中。

如果分区数据库环境中的一个或多个数据库分区遇到错误，那么返回的 SQL 代码是来自报告错误的第一个节点的 SQL 代码。

如果运行时出现故障，那么联机表重组将暂停并进行回滚。如果系统宕机，那么在重新启动时，将开始进行崩溃恢复，并且会暂停并回滚重组。稍后，您可以通过使用 REORG TABLE 命令并指定 RESUME 选项来继续重组。由于完整地记录了联机表重组过程，因此可以保证重组过程可恢复。

例如，在某些情况下，联机表重组可能超过 num_log_span 限制。在这种情况下，DB2 将强制执行表重组并使其处于暂停状态。在快照输出中，REORG TABLE 命令的状态将显示为“已暂停”。

联机表重组暂停是由输入驱动的，这意味着它可以由用户(通过使用 REORG TABLE 命令的暂停(pause)选项或强制执行应用程序命令)或 DB2 在某些情况下(例如系统崩溃时)暂停。

10. 暂停并重新启动联机表重组

用户可以暂停并重新启动正在进行的联机表重组。

要暂停联机表重组，请发出使用 PAUSE 选项的 REORG TABLE 命令：

```
db2 reorg table homer.employee inplace pause
```

要重新启动已暂停的联机表重组，请发出带有 RESUME 选项的 REORG TABLE 命令：

```
db2 reorg table homer.employee inplace resume
```

注意：

- 暂停联机表重组后，不能开始对该表进行新的重组。必须继续暂停的重组，或者停止暂停的重组，然后再开始新的重组过程。
- 发出 RESUME 请求后，重组过程将沿用在原始 REORG 命令中指定的 TRUNCATE 选项，而无论对后续 START 或任何中间 RESUME 请求指定什么 TRUNCATE 选项。但是，如果重组处于截断阶段并且用户发出指定了 NOTRUNCATE 的 RESUME 请求，那么不会截断表并且重组将完成。
- 在复原和前滚操作后，重组不能继续。

11. 联机表重组的锁定和并行性注意事项

在做联机表重组时，需要考虑的一个重要方面是如何控制锁定，因为它对于应用程序的并行性非常重要。

在联机表重组过程中的某个给定时间点，操作可能拥有下列锁定：

- 为了确保能够对表空间进行写访问，获取对重组表操作所影响的表空间的 IX 锁定。
- 在整个重组表操作期间获取并挂起表锁定。锁定级别取决于重组期间允许对该表使用的访问方式：
  - ◊ 如果指定了 ALLOW WRITE ACCESS，那么将获取对表的 IS 锁定。
  - ◊ 如果指定了 ALLOW READ ACCESS，那么将获取对表的 S 锁定。
- 在截断阶段，当重组操作将行移出截断范围时，将请求对表的 S 锁定，因为旧扫描程序(重组表操作期间存在并访问记录的旧 RID 的数据访问)可能会插入新行。DB2 将等到获取此锁定后再开始截断。就在截断阶段结束时重组表操作对表进行物理截断的那一刻，S 锁定将升级为特殊的 Z 锁定。这表示直到任何现有应用程序都不拥有表锁定(包括来自 UR 扫描程序的 IN 锁定)时，重组表操作才完成。
- 还可以根据表锁定类型获取行锁定：
  - ◊ 如果拥有对表的 S 锁定，那么不需要单独的行级别 S 锁定，因此不需要进一步锁定。
  - ◊ 如果拥有对表的 IS 锁定，那么在移动行之前将获取行级别 S 锁定，并在移动完成后释放该锁定。
- 可能需要内部锁定来控制对联机表重组对象和其他联机 DB2 实用程序(如联机备份)的访问。

锁定会影响重组表和并发用户应用程序的性能。强烈建议您在执行联机表重组时检查

锁定快照数据以了解锁定活动。

12. 监视表重组

可以使用GET SNAPSHOT命令、SNAPTAB_REORG管理视图或SNAP_GET_TAB_REORG表函数获取有关表重组操作的状态的信息。

要使用 SQL 访问有关表重组操作的信息，请使用 SNAPTAB_REORG 管理视图。例如，以下 select 语句返回有关在当前连接的数据库上对所有数据库分区执行的表重组操作的详细信息：

```
SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA, 1, 15)
   AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,
   REORG_STATUS, REORG_COMPLETION, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPTAB_REORG ORDER BY DBPARTITIONNUM
TAB_NAME   TAB_SCHEMA      REORG_PHASE      REORG_TYPE      REORG_STATUS REORG_COMPLETION
--------------- --------------- ---------------- -------------------- ------------
----------------
ACCOUNT          ORACLE          REPLACE          RECLAIM+OFFLINE+ALLO COMPLETED    SUCCESS
  1 条记录已选择。
```

如果未重组任何表，那么返回 0 行。

要使用快照监视器访问有关表重组操作的信息，请发出 GET SNAPSHOT FOR TABLES ON database 命令并检查表重组监视元素(具有前缀“reorg_”)的值。

由于脱机表重组是同步的，所以脱机表重组中出现的任何错误都会返回给实用程序的调用者(应用程序或命令行)。

联机表重组中的错误处理与脱机表重组中的错误处理略有不同。由于联机表重组是异步的，所以没有 SQL 消息写入 CLP。要查看在联机表重组期间返回的 SQL 错误，请发出“LIST HISTORY REORG”命令。

联机表重组作为 db2reorg 进程在后台运行。这意味着从重组表进程成功返回到调用应用程序并不表示重组已成功完成。即使调用应用程序终止其数据库连接，db2reorg 进程也会继续。

11.3.2 索引重组

通过删除和插入操作对表进行更新后，索引的性能会降低，其表现方式如下：

- 索引叶子页分裂
- 叶子页被分裂之后，由于必须读取更多的叶子页才能访存表页，因此 I/O 操作成本会增加。

- 物理索引页的顺序不再与这些页上的键顺序相匹配(此称为不良集群索引)。
- 叶子页出现不良集群情况后，顺序预取操作的效率将降低，因此会导致更多的 I/O 等待。
- 形成的索引大于其最有效的级别(level)数。

在此情况下应重组索引。

如果在创建索引时设置了 MINPCTUSED 参数，那么在删除某个键且可用空间小于指定的百分比时，数据库服务器会自动合并索引叶子页。此过程称为联机索引整理碎片。但是，要复原索引集群和可用空间以及降低索引叶级别，请使用下列其中一种方法：

- 删除并重新创建索引。
- 使用 REORG INDEXES 命令联机重组索引。因为此方法允许用户在重建表索引期间对表进行读写操作，所以在生产环境中可能需要选择此方法。
- 使用允许脱机重组表及其索引的选项运行 REORG TABLE 命令。

联机索引重组

在使用 ALLOW WRITE ACCESS 选项运行 REORG INDEXES 命令时，如果同时允许对指定的表进行读写访问，那么会重建该表的所有索引。进行重组时，对基表所作的任何将会影响到索引的更改都将记录在 DB2 日志中。另外，如果有任何内部内存缓冲区空间可供使用，那么还将这些更改放在这样的内存空间中。重组将处理所记录的更改以便在重建索引时与当前写活动保持同步更新。内部内存缓冲区空间是根据需要从实用程序堆中分配的指定内存区域，它用来存储对正在创建或重组的索引所作的更改。使用内存缓冲区空间使索引重组操作能够通过这样的方式来处理更改，即先直接从内存读取，然后读取日志(如有必要)，但读取日志的时间要晚得多。在重组操作完成后，将释放所分配的内存。重组完成后，重建的索引可能不是最佳集群的索引。如果为索引指定 PCTFREE，那么在重组期间，每页上均会保留相应百分比的空间。

对于分区表，支持对各个索引进行联机索引重组和清除。要对各个索引进行重组，指定索引名：REORG INDEX *index_name* for TABLE *table_name*

注意：

REORG INDEXES/INDEX 命令的 CLEANUP ONLY 选项不能完全重组索引。CLEANUP ONLY ALL 选项将除去那些标记为“删除”且被认为要落实的键。此外，它还将释放所有标记为“删除”且被认为要落实的键所在的页。在释放页后，相邻的叶子页将会合并，前提是这样做可以在合并页上至少留出 PCTFREE 可用空间。PCTFREE 是指在创建索引时为其定义的可用空间百分比。CLEANUP ONLY PAGES 选项仅删除那些标记为“删除”且被认为要落实的所有键所在的页。

使用 CLEANUP ONLY 选项对分区表的索引进行重组时，支持任何访问级别。如果未指定 CLEANUP ONLY 选项，那么默认访问级别 ALLOW NO ACCESS 是唯一支持的访问级别。

索引重组具有下列要求：

- 对索引和表具有 SYSADM、SYSMAINT、SYSCTRL 或 DBADM 权限，或者具有 CONTROL 特权。
- 用于存储索引的表空间的可用空间数量大于或等于索引的当前大小。在发出 CREATE TABLE 语句时，考虑在大型表空间中重组索引。
- 其他日志空间。索引重组需要记录其活动。因此，重组可能会失败，尤其是在系统繁忙和记录其他并发活动时。

注意：

如果具有 ALLOW NO ACCESS 选项的 REORG INDEXES ALL 命令运行失败，那么会标记索引无效并且此项操作不可撤销。但是，如果具有 ALLOW READ ACCESS 选项的 REORG 命令或具有 ALLOW WRITE ACCESS 选项的 REORG 命令运行失败，那么可以复原原来的索引对象。

11.3.3 确定何时重组表和索引

在对表数据进行许多更改之后，逻辑上连续的数据可能会位于不连续的物理数据页上，在许多更新操作创建了溢出(overflow)记录时尤其如此。按这种方式组织数据时，数据库管理器必须执行额外的读操作才能访问顺序数据。另外，在删除大量数据后，空间并没有释放，也需要执行额外的读操作。

表重组操作会通过整理数据碎片来减少浪费的空间，并对行进行重新排序以合并溢出记录，从而加快数据访问速度并最终提高查询性能。还可以指定根据特定索引来重新排序数据，以便查询通过最少的 I/O 读取操作就可以访问数据。

对表数据进行大量更改将导致更新索引并使索引性能下降。索引叶子页可能变成碎片或出现不良集群情况，并且索引有可能形成比所需层次(level)要多的层次以获得最佳性能(通常，几百万条记录的索引层次一般为 3，正常生产环境中索引的层次很少超过 4。)。所有这些问题都会产生更多 I/O 并导致性能下降。

下列任何情况都需要我们重组表或索引：

- 自上次重组表之后，对查询所访问的表进行了大量的插入、更新和删除活动
- 对于使用具有高聚合度的索引的查询，其性能发生了明显变化
- 在执行 RUNSTATS 以刷新统计信息后，性能没有得到改善

- REORGCHK 命令指示需要重组表或索引(注意：在某些情况下，REORGCHK 始终建议重组表，即使在执行了重组后也是如此)。例如，如果使用 32KB 页大小，并且平均记录长度为 15 字节，每页最多包含 253 条记录，那么每页具有 32700－(15×253)=28905 个不可用字节。这意味着大约 88%的页面是可用空间。用户应分析 REORGCHK 的建议并针对执行重组所需的成本平衡利益。
- 综合考虑查询性能不断降低所浪费的成本和重组表所需的成本(包括 CPU 时间、重组的时间和 REORG 实用程序在完成重组操作之前锁定表造成的并行性降低)，以确定是否进行表重组。

要确定是否需要重组表或索引，查询系统目录表中的统计信息并监视下列统计信息：

行的溢出(行链接)

查询 SYSSTAT.TABLES 视图中的 OVERFLOW 列以监视溢出值。当表中的可变长度列导致记录长度变长，以致它们不能放入数据页上的指定位置时，这时行数据就会溢出。在列添加至表定义并稍后通过更新行来更新该列时，长度也可能会更改。在这些情况下，在行中的原始位置将保留一个指针，而实际值存储在由指针指示的另一个位置。这可能会影响性能，因为数据库管理器必须根据指针来查找行的内容。这个包括两个步骤的过程增加了处理时间，并且还会增加需要的 I/O 数目。

重组表数据将消除行溢出，如果行链接数量较多，重组表数据的效果就会比较明显。

访存统计信息

当以索引顺序访问表时，查询 SYSCAT.INDEXES 和 SYSSTAT.INDEXES 系统目录表中的以下 3 个列来确定预取程序的效率。这些统计信息对照基表来体现预取程序的平均性能的特征。

- AVERAGE_SEQUENCE_FETCH_PAGES 列存储可以按表中顺序访问的平均页数。可以按顺序访问的页适合于预取。较小的数目指示预取程序没有充分发挥作用，原因是它们不能读入由表空间的 PREFETCHSIZE 设置指定的所有页数。较大的数目指示预取程序将有效地执行。对于集群索引和表，此数目应该接近 NPAGES(包含一些行的页数)的值。
- AVERAGE_RANDOM_FETCH_PAGES 列存储当使用索引来访存表行时，在顺序页访问之间的平均随机表页数。当大多数页按顺序排列时，预取程序忽略少量的随机页，并按已配置的预取大小继续预取。当表变得更加混乱时，随机访存页数就会增加。通常，由于在表的末尾或在溢出页中发生了无序的插入，导致了这样

的表的无组织。当使用索引来访问某一范围内的值时，这会导致降低查询性能的访存。

- AVERAGE_SEQUENCE_FETCH_GAP 列存储当使用索引进行访存时表页序列之间的平均间隔。通过扫描索引叶子页来进行检测，每个间隔表示在表页的各个序列之间必须随机访存的平均表页数。当随机访问许多页时发生这些情况，这会中断预取程序。较大的数指示无组织的表或较差集群的索引。

包含标记为已删除但未除去的 RID 的索引叶子页数

在 type-2 类型索引中，当 RID 标记为删除时，通常未在物理上删除 RID。这意味着可用空间可能会被这些逻辑上已删除的 RID 占用。要检索每个 RID 都被标记为已删除的叶子页的数目，查询 SYSCAT.INDEXES 和 SYSSTAT.INDEXES 统计信息表的 NUM_EMPTY_LEAFS 列。对于并非所有 RID 都标记为删除的叶子页，逻辑上已删除的 RID 的总数存储在 NUMRIDS_DELETED 列中。

使用此信息来通过执行带 CLEANUP ALL 选项的 REORG INDEXES 估计可以回收多少空间。要想只回收所有 RID 都被标记为已删除的页中的空间，执行带有 CLEANUP ONLY PAGES 选项的 REORG INDEXES。

索引的聚合比率和聚合因子统计信息

聚合比率统计信息存储在 SYSCAT.INDEXES 系统目录表的 CLUSTERRATIO 列中。此值(在 0 到 100 之间)表示使用索引的数据聚合的程度。如果收集 DETAILED 索引统计信息，0 和 1 之间的好的聚合因子统计信息存储在 CLUSTERFACTOR 列中，并且 CLUSTERRATIO 的值为－1。这两个集群统计信息中只有一个可以记录在 SYSCAT.INDEXES 目录表中。要将 CLUSTERFACTOR 值与 CLUSTERRATIO 值进行比较，可以将 CLUSTERFACTOR 乘以 100 以获得一个百分比。

注意：
通常，表中只有一个索引可以具有较高的聚合度。

如果查询用到的索引需要访问表中比较多的数据，而不是只访问索引或者只通过唯一索引访问表的一条记录，那么在具有较高聚合比率的情况下可能执行得更好。低的聚合度导致此类扫描要执行更多的 I/O，因为在每个数据页经过第一次访问后，下次访问该页时，该页仍在缓冲池中的可能性减小。增大缓冲区大小也可以提高非聚合索引的性能。

如果表数据最初是对某个索引聚合的，而集群统计信息指示现在很少为同一索引聚合数据，那么您可能想重组该表以再次聚合数据。

索引叶子页数

查询 SYSCAT.INDEXES 表中的 NLEAF 列以便了解索引占用的叶子页数目。该数目告诉您对索引进行完整的扫描需要多少索引页 I/O。

理想情况下，索引应该尽可能占用最小的空间量，以便减少进行索引扫描所需要的 I/O 次数。随机的更新活动可导致页分割，这就增大了索引的大小。当在重组表期间重建索引时，可以使用最小空间量来构建每个索引。

注意：

默认情况下，当构建索引时，会在每个索引页上保留 10%的可用空间。要增加可用空间量，当创建索引时指定 PCTFREE 参数。无论何时重组索引，都使用 PCTFREE 值。

空数据页的数目

要计算表中的碎片数，查询 SYSCAT.TABLES 中的 FPAGES 和 NPAGES 列并用 FPAGES 值减去 NPAGES 值。FPAGES 列存储正在使用的总页数；NPAGES 列存储包含一些行的页数。当删除整个范围内的行时，可能出现空页。

随着碎片的增多，就更需要进行表重组。重组表时将回收碎片并减少表所使用的空间量。另外，因为会将碎片的数据页读入缓冲池以进行表扫描，所以回收未使用的页可以提高表扫描的性能。

REORGCHK 命令返回有关数据组织的统计信息，并且可以在是否需要重组特定表或索引这一问题上为您提供建议。以下为 RECORGCHK 的输出。

```
C:\pp>db2 reorgchk update statistics
正在执行 RUNSTATS ....
表统计信息：
F1: 100 * OVERFLOW / CARD < 5
F2: 100 *(数据页的有效空间使用率) > 70
F3: 100 *(需要页数 / 总页数) > 80
SCHEMA.NAME          CARD    OV    NP    FP ACTBLK   TSIZE  F1  F2  F3 REORG
表：ORACLE.ACCOUNT
                   100000      0  1820  1820      - 7200000   0  98 100 ---
表：ORACLE.AUDITLOG
                     3382      0    29    29      -  114988   0 100 100 ---
表：ORACLE.CONNHEADER_T1
                         21      0     2     2      -    4158   0 100 100 ---
表：ORACLE.CONTROL_T1
                         2      0     1     1      -      84   0   - 100 ---
表：ORACLE.STMT_T1
```

```
                        403      0     58     58      -   205127    0 100 100 ---
    表：SYSTOOLS.HMON ATM INFO
                        136     86      6     13      -    47872   63 100 100 *--
    表：SYSTOOLS.HMON COLLECTION
                          -      -      -      -      -        -    -   -   - ---
    表：SYSTOOLS.POLICY
                          5      0      1      1      -      740    0   - 100 ---
```

在上面的输出中，我们重点关注“REORG”字段，如果 REORG 字段有一个或多个“*”，那么可以考虑对该表或索引重组。

在重组之前，请综合考虑查询性能不断降低所浪费的成本和重组表或索引所需的成本(包括 CPU 时间、耗用时间和 REORG 命令在完成重组操作之前锁定表造成的并行性降低)，以确定是否进行表重组。

11.3.4　重组表和索引的成本

决定是否要重组表或索引时，必须考虑对这些对象执行重组所产生的一些开销。

重组表和索引的成本包括：

- 执行 REORG 命令时消耗的 CPU。
- 运行 REORG 命令时并行性降低。由于重组要求进行锁定而导致并行性降低。
- 需要额外的存储空间(脱机表重组需要额外存储空间来保存表的影子副本。联机表重组需要更多的日志记录空间。联机索引重组需要额外存储空间来保存索引的影子副本和更多日志空间。脱机索引重组将使用较少的日志空间并且不涉及影子副本)。

在某些情况下，已重组的表可能比原始表要大，从而相应地增加了空间要求。在下列情况下，重组后表可能会增大：

- 在使用索引来确定行顺序的集群重组表中，如果表记录的长度可变(例如，使用 varchar)，那么重组结束时您可能会使用更多空间，因为某些页中包含的行数可能比原始表中的行数要少。
- 在重组之前但在创建表之后在表中添加了列，并且重组之后在某些行中可能是第一次实现添加的列。
- 自上次重组后每页上剩余的可用空间大小(由 PCTFREE 属性的值表示)已增加。
- 表包含 LOB，并且该 LOB 可能比以前使用更多的空间。

脱机表重组的空间要求

由于脱机表重组使用影子副本方法，因此需要足够的额外存储空间来容纳表的另一个副本。在原始表所在的表空间中或用户指定的临时表空间中构建影子副本。

如果使用表扫描排序，那么可能需要其他临时表空间来进行排序处理。需要的其他空间可能与要重组的表一样大。任何其他需要重新创建的索引都将涉及排序，并可能需要多达要重组的表大小的临时表空间。

脱机表重组生成少量控制日志记录，因此消耗相对较少的日志空间。如果重组未使用索引，那么唯一的日志记录是表数据日志记录。如果指定了索引，或者表上存在集群索引，那么按照将行放入新版本的表中的顺序记录这些行的 RID。每条 RID 日志记录最多包含 8000 个 RID，每个 RID 消耗 4 字节。这可能是导致在脱机表重组期间用完日志空间的一个因素。请注意，仅当数据库可恢复(LOGRETAIN=ON)时，才记录 RID。

联机表重组的日志空间要求

联机表重组所需的日志空间通常比脱机表重组所需的日志空间要大。需要的空间大小由要重组的行数、索引数、索引键的大小，以及最开始表的组织情况决定。为用于表的日志空间确定一个典型的基准是一种不错的做法。

表中的每行都可能在联机表重组期间移动两次。如果提供一个索引，那么每行必须更新索引键以反映新位置，并且在完成所有对旧位置的访问后，将再次更新索引键以除去对旧 RID 的引用。移回行时，将再次执行对索引键的这些更新。将记录所有这些活动以便联机表重组完全可恢复，因此最少应有 2 条数据日志记录(包括行数据的每个实例)和 4 条索引日志记录(包括键数据的每个实例)。集群索引尤其容易填满索引页，从而导致索引分裂和合并，这些分裂和合并活动也必须进行记录。

由于联机表重组经常发出内部落实，所以它通常不会拥有重要的活动日志。如果在某个时间联机重组拥有大量活动日志，那么必定是在截断阶段，因为截断阶段需要 S 表锁定。如果表重组无法获取该锁定，那么它将等待并保留日志，而其他事务可能会快速填满日志。

11.3.5　合理设计以减少碎片生成

您可以使用不同的策略来降低需要重组表和索引的频率，从而避免执行不必要的重组操作的成本。

减少重组表的需要

- 创建多维集群(MDC)表，将在 CREATE TABLE 语句的 ORGANIZE BY DIMENSION 子句中所指定的列中自动维护该表的集群。
- 对表打开 APPEND 方式。例如，如果这些新行的索引键值总是新的更大的键值，那么表的集群属性将尝试将其放置在表的末尾。在这种情况下，将表置于追加方式(append on)可能优于集群索引。
- 在创建表之后：

◊ 改变表以添加 PCTFREE
◊ 在索引上使用指定的 PCTFREE 来创建集群索引。
◊ 将数据装入到表中之前对数据进行排序

- 装入数据

表上正确设置了 PCTFREE 的集群索引有助于保持原始排序顺序。当表页上有足够的空间时，可以将新数据插入正确的页，以维护索引的集群特征。随着更多的数据插入，表页会因此而变满，记录会被追加至表的末尾，因而表将逐渐失去集群特性。

如果在创建集群索引后执行 REORG TABLE 命令，或者执行排序和 LOAD 命令，那么索引会尝试维护数据的特定顺序，这将改善 RUNSTATS 实用程序所收集的 CLUSTERRATIO 或 CLUSTERFACTOR 统计信息。

注意：

创建多维集群(MDC)表可以减少重组表的必要性。对于 MDC 表，将在 CREATE TABLE 语句的 ORGANIZE BY DIMENSIONS 子句中指定为参数的列上维护集群。

减少重组索引的需要

- 在索引页上使用 PCTFREE 或 LEVEL2 PCTFREE 创建集群索引。范围在 0 到 99% 之间，默认值为 10%。
- 使用 MINPCTUSED 创建索引。可能的范围在 0 到 99%之间，建议的值为 50%。
- 考虑使用 REORG INDEXES 命令的 CLEANUP ONLY ALL 选项来合并叶子页。

11.3.6 启用表和索引的自动重组

在 DB2 V8 之前，我们对表和索引做碎片整理，只能通过写脚本方式定期在业务逻辑许可的情况下对表和索引做碎片整理。DB2 V8 之后可以使用自动表重组来启用 DB2 管理脱机和索引重组，以使您无须担忧何时使用何种方法来重组数据。可启用 DB2 来重组系统目录表以及数据库表。

可以使用图形用户界面工具或命令行界面来打开此功能。

- 要使用图形用户界面工具来将数据库设置为自动重组：

(1) 通过以下两种方法之一来打开“配置自动维护”向导：一种方法是在“控制中心”中右击数据库对象；另一种方法是在“运行状况中心”中右击要配置自动重组的数据库实例。从弹出窗口中选择**配置自动维护**。

(2) 在此向导中，可以启用自动重组来整理数据碎片，指定想要自动重组的表，以及指定用于执行 REORG 实用程序的维护窗口。

- 要使用命令行界面来将数据库设置为自动重组，请将下列配置参数设置为“ON”：
  - ◊ AUTO_MAINT
  - ◊ AUTO_TBL_MAINT
  - ◊ AUTO_REORG

不过我不建议您使用自动表重组，因为自动表重组是数据库控制的，我们不能干预，它很可能在业务高峰期间开始了自动表重组，这会影响性能。所以，我建议在充分了解业务逻辑的情况下，通过写 crontab 脚本在合适的时间来调度执行碎片整理。

11.4　碎片整理案例

11.4.1　执行表、索引检查是否需要做 REORG

首先执行表、索引检查是否需要做碎片整理：

```
--------------------------------------------------------
--DB2 CLP
--------------------------------------------------------
db2 reorgchk update statistics on table db2admin.employee
    执行 RUNSTATS ....
    表统计信息:
F1: 100 * OVERFLOW / CARD < 5
F2: 100 *(Effective Space Utilization of Data Pages) > 70
F3: 100 *(Required Pages / Total Pages) > 80
SCHEMA     NAME      CARD     OV      NP      FP ACTBLK     TSIZE  F1   F2   F3 REORG
-----------------------------------------------------------------------------------
DB2ADMIN  EMPLOYEE       258500 51699 12932 16165    - 61781500  19  93  80 *-*
-----------------------------------------------------------------------------------
索引统计信息:
F4: CLUSTERRATIO 或正常化的 CLUSTERFACTOR > 80
F5: 100 *(KEYS *(ISIZE + 9) +(CARD - KEYS) * 5) /((NLEAF - NUM EMPTY LEAFS)
    * INDEXPAGESIZE) > 50
F6:(100 - PCTFREE) *((INDEXPAGESIZE - 96) /(ISIZE + 12)) **(NLEVELS - 2)
    *(INDEXPAGESIZE - 96) /(KEYS *(ISIZE + 9) +(CARD - KEYS) * 5) < 100
F7: 100 *(NUMRIDS DELETED /(NUMRIDS DELETED + CARD)) < 20
F8: 100 *(NUM EMPTY LEAFS / NLEAF) < 20
SCHEMA    NAME    CARD  LEAF ELEAF  LVLS ISIZE  NDEL  KEYS  F4  F5  F6  F7  F8 REORG
-----------------------------------------------------------------------------------
表: DB2ADMIN.EMPLOYEE
DB2ADMIN IDX EMP C   258500 14894  0    4  106 21040 258500  72  48  13   7   0 **---
SYSIBM   SQL060417152213950 258500  7122      0       4     60       0
```

```
258500  72  61  62   0   0 *----
```

CLUSTERRATIO 或正常化的 CLUSTERFACTOR(F4)将指示索引需要 REORG，该索引与基本表不在相同的序列中。当在表中定义了多个索引时，一个或多个索引可能被标记为需要 REORG。指定 REORG 顺序的最重要索引。

使用 ORGANIZE BY 子句和相应的维索引定义的表的名称有'*'后缀。维索引的基数等价于表的“活动的块数”统计信息。

DB2 提示信息说明

对 REORGCHK 所使用的度量的考虑因素包括(当查看 REGORGCHK 工具的输出时，找到用于表的 F1、F2 和 F3 这几列，以及用于索引的 F4、F5、F6、F7 和 F8 这几列。如果这些列中的任何一列有星号(*)，则说明当前的表和/或索引应该重组)：

F1：属于溢出记录的行所占的百分比。当这个百分比大于 5% 时，在输出的 F1 列上将有一个星号(*)。

F2：数据页中使用了的空间所占的百分比。当这个百分比小于 70%(也就是说有 30%左右的碎片) 时，在输出的 F2 列上将有一个星号(*)。

F3：其中含有包含某些记录的数据的页所占的百分比。当这个百分比小于 80% 时，在输出的 F3 列上将有一个星号(*)。

F4：群集率，即表中与索引具有相同顺序的行所占的百分比。当这个百分比小于 80%时，在输出的 F4 列上将有一个星号(*)。

F5：在每个索引页上用于索引键的空间所占的百分比。当这个百分比小于 50% 时，在输出的 F5 列上将有一个星号(*)。

F6：可以存储在每个索引级的键的数目。当这个数字小于 100 时，在输出的 F6 列上将有一个星号(*)。

F7：在一个页中被标记为 deleted 的记录 ID(键)所占的百分比。当这个百分比大于 20%时，在输出的 F7 列上将有一个星号(*)。

F8：索引中空叶子页所占的百分比。当这个百分比大于 20%时，在输出的 F8 列上将有一个星号(*)。

11.4.2 表和索引碎片整理

(1) 针对 reorgchk 给出的提示信息(特别是打*号的 REORG 列)，结合 SQL 语句本身的构成，建立适当的索引。

(2) 根据实际情况，重组表、重组索引。

(3) 更新表、索引统计信息。

(4) 如果应用程序是静态 SQL，重新绑定(rebind)应用程序包。

例如：

```
--------------------------------------------------------
--DB2 CLP
--------------------------------------------------------
db2 reorg table db2admin.employee;
db2 reorgchk update statistics on table db2admin.employee;
db2 reorg indexes all for table db2admin.employee;
db2 runstats on table db2admin.employee and indexes all;
```

11.5　案例：生成碎片检查、统计信息更新、碎片整理和 REBIND 脚本

在实际生产中，我们可以编写统计信息更新、碎片整理和重新绑定脚本来调度统计信息更新，脚本如下：

```
    db2 connect to sample
    db2 “select 'reorg  table '||rtrim(tabschema||'.'||tabname||';' from
syscat.tables where type='T' > reorg.sql
    db2 “select 'reorg indexes all for table
'||rtrim(tabschema||'.'||tabname||';' from syscat.tables where type='T' >
reorg_index.sql
    db2 “select 'runstats on table '||rtrim(tabschema||'.'||tabname||' and
indexes all;' from syscat.tables where type='T' > runstats.sql
    db2 “select 'rebind package '||rtrim(pkgschema||'.'||pkgname||';' from
syscat.packages where pkgschema not in('NULLID' ) > rebind.sql
```

把生成的结果分别编辑，存放到特定目录下。

在 UNIX 上编写 shell 脚本 maint.sh：

```
While 1>0 do
db2 -tvf reorg.sql -z reorg_error.log
db2 -tvf reorg_index.sql -z reorg_index.log
db2 -tvf runstats.sql -z runstats_error.log
db2 -tvf rebind.sql -z reinbd_error.log
```

然后用 crontab 调度在合适的时间(业务最闲的时候)执行，例如：

```
#每天早上 3 点执行一次 /bin/ls :  0 3 * * * /home/db2inst1/sqllib/mon/maint.sh
```

11.6 重新绑定程序包

重新绑定(REBIND)是为先前绑定的应用程序重新创建程序包的过程。每个在数据库中执行的应用程序在执行之前都需要有一个绑定过程，这个绑定过程会根据数据库中各种统计信息和相关数据库对象的情况创建一个程序包，这个程序包中通常就是执行计划。所以在统计信息或相关数据库对象被修改之后，就需要对数据库中受影响的应用程序执行重新绑定，这样才能允许应用程序只用最新的更新。

那么具体的哪些程序包必须要执行重新绑定呢？DB2 的系统表 SYSCAT.PACKAGES 的 VALID 列标识当前的程序包是否可用，如果此列的值为 X，则表示当前的程序包是不可用的，此程序包就需要被重新绑定。这些程序包可以使用下面的方法进行绑定，也可以让数据库管理器在执行这样的程序包的时候自动地完成重新绑定(数据库管理器默认会在隐含执行这些不可用的程序包的时候重新将其绑定)。

建议在对统计信息更新之后，对数据库中的静态应用程序包执行重新绑定，以更新执行计划。还有一些其他情况需要及时地重新绑定程序包，如创建了新的索引等。统计信息更新、碎片整理和 REBIND 的顺序如图 11-2 所示。

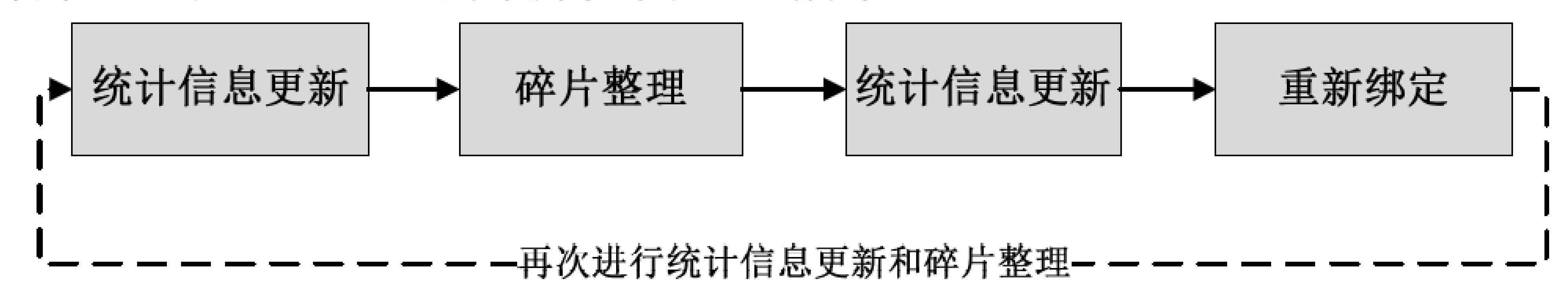


图 11-2　统计信息更新和碎片整理的处理流程

有两个命令可以执行重新绑定：

- rebind
- db2rbind

rebind 命令

rebind 命令可以实现重新创建一个数据库中已经存在的程序包。具体的语法如下：

```
>>-REBIND--+---------+--package-name---------------------------->
           '-PACKAGE-'
>--+----------------------+--RESOLVE--+-ANY----------+--------->
   '-VERSION--version-name-'          '-CONSERVATIVE-'
```

```
>--+--------------+--------------------------------------------><
   +-REOPT NONE---+
   +-REOPT ONCE---+
   '-REOPT ALWAYS-'
```

db2rbind 命令

db2rbind 命令可以重新绑定数据库中存在的所有的程序包。具体语法如下：

```
>>-db2rbind--database-- -l--logfile--+-----+------------------->
                                     '-all-'
>--+---------------------------+--+----------------------+---><
   '- -u--userid-- -p--password-'  |      .-conservative-. |
                                   '- -r--+-any----------+-'
```

此命令的功能类似于 REBIND 命令，只是更简单些，而且可以批量地处理数据库中所有的程序包。

11.7　数据库运行维护总结

当数据库运行了较长时间之后，随着数据的变化或增长，数据库中的应用会变得越来越慢，我们就不得不对数据库进行统计信息更新、碎片整理和 rebind 的工作以提高应用的性能。而且通常会一起考虑执行。并且应该有图 11-3 所示的处理流程：

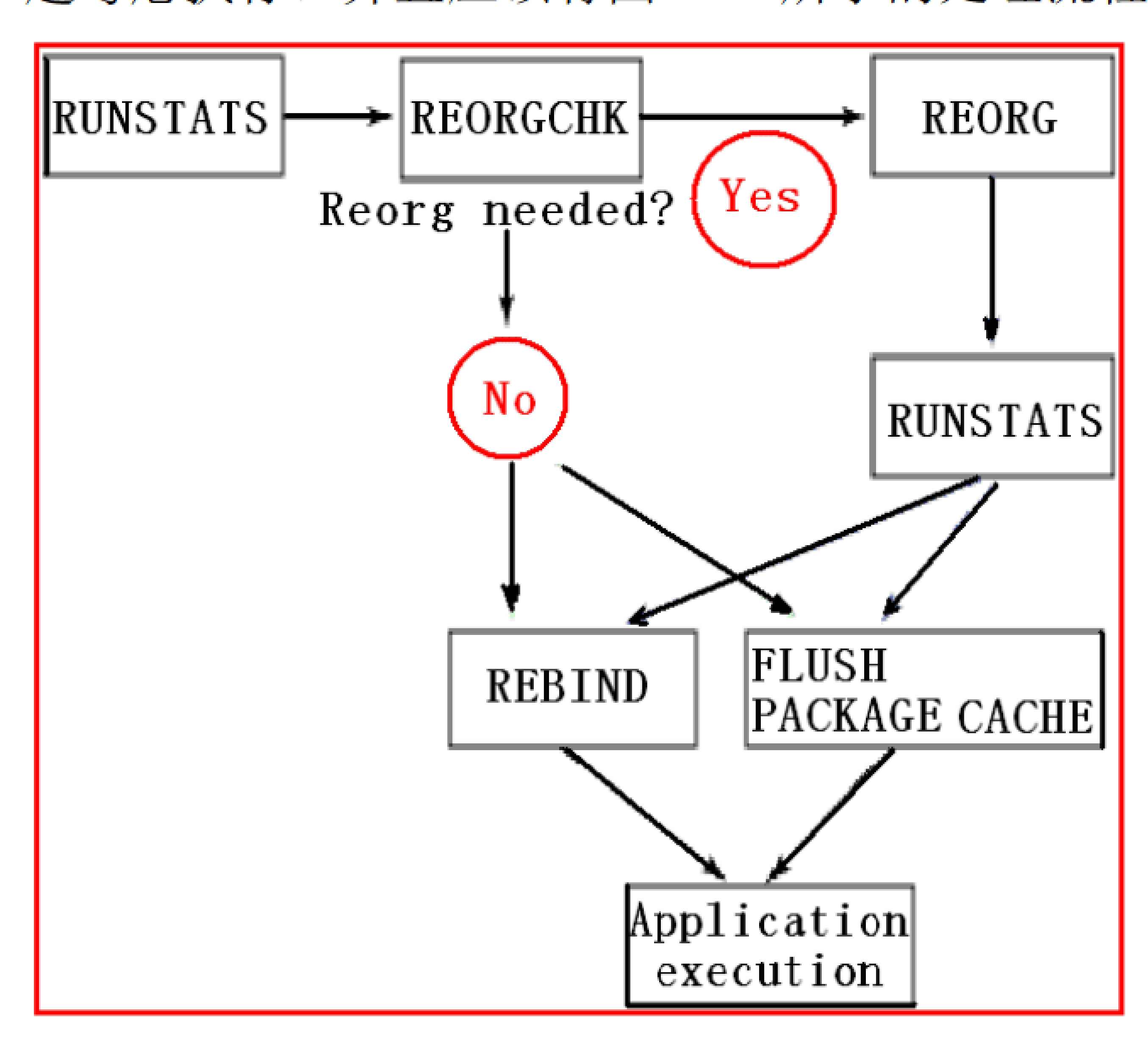


图 11-3　统计信息更新和碎片整理的处理流程

首先使用统计信息更新工具 RUNSTATS 更新相关数据库对象的统计信息，然后使用数据库重组工具 REORG 对数据库的碎片进行处理，接着再使用 RUNSTATS 工具对数据库的统计信息更新，最后将数据库中相关的程序包重新绑定。这样就完成了一个完整的统计更新和碎片整理的过程。在经过一定的时间周期后(一周、一个月或其他时间)，再次完成上面所描述的完整过程。**这样在数据库中可以访问数据时，不会由于统计信息误差过大而选择成本高昂的访问策略。**

那么对于上述的循环过程的执行频率应该是何种频率呢？答案是：不一定。这需要根据实际情况来确定。最重要的就是我们要明确上述过程的最终目标是什么？上述过程的最终目标就是将数据库中数据的变化反映出来并让数据库作出相应的调整。所以数据库中数据变化得越快，那么上述过程的频率就应该越高；相反地，数据库中数据变化得越慢，那么上述过程的频率也就越低。如果数据库中的数据没有变化，我们甚至可不需要执行上述的过程。还有个需要关注的因素就是时间窗口，实际的应用程序是否能够为我们执行上述过程留出足够的时间窗口。比如，应用是一个 5×24 的应用，那么平时是没有时间窗口让我们来完成上述过程的，这些工作只能放到周末去执行。还有，假设有一个非常大的表，如果通过常规的方法：碎片检查，表和索引碎片整理，最后统计信息更新。这个可能要经过很长的时间。如果这个表不是 7×24 的应用。那我建议你最快的方法是把该表的数据导出(export)，然后用带 statistics 选项的 load replace 操作来转载数据。这样不但做了表和索引的碎片整理，同时还做了统计信息更新。而且时间比常规方式会大大缩短。总而言之，最终采用何种频率、何种方式来完成上面过程，就是要考虑到上面的两方面因素来做决定。



数据库常用工具

DB2 数据库为我们提供了很多功能强大的工具，这些工具能够帮助我们解决某一特定的问题。本章主要讲解 DB2 中一些最常用的工具。

本章主要讲解如下内容：

- 解释工具
- 索引设计工具
- 基准测试工具
- 数据一致性检查工具
- db2look 工具
- 其他工具

12.1 解释工具

实际运行中，应用中的某些 SQL 语句往往比较消耗资源，当我们使用监控工具定位出特定的 SQL 语句时，我们需要对该 SQL 语句做解释分析，这就需要用到解释工具。DB2 提供了可视化解释、db2expln、dynexpln 和 db2exfmt 等几种常用工具。

12.1.1 Visual Explain(可视化解释)

Visual Explain 是一种 GUI 工具，它为数据库管理员和应用程序开发人员提供了查看为特定 SQL 语句选择的访问计划的图形化表示的能力。但 Visual Explain 只能用于查看解释快照数据或人工输入 SQL 或脚本，要查看已收集并写入了解释表的全面解释数据，则必须使用 db2exfmt 或 db2expln 工具。

1. timeron

为了分析解释信息，您需要了解的最重要的事情就是 timeron 这一概念。timeron 是 DB2

优化器使用的一种成本度量单位，用于计算查询完全执行所需的时间和资源数量。timeron是时间、CPU 占用率、磁盘 I/O 和其他一些因素的综合。由于这些参数的值不断变化，执行一个查询所需的 timeron 数量也是动态的，每次执行都有所不同。

timeron 也是一种创造出来的度量单位，因此，没有什么公式可以将执行一个查询所需的 timeron 数转换成秒数。除此之外，timeron 可以帮助您确定一种查询执行途径是否比另一种更快(如果执行一个查询所需的 timeron 数在两次编译之间相差 10 或 20，那么不必担心，因为这可能仅仅是由于 CPU 活动、磁盘活动或数据库使用情况发生了变化造成的)。

2. SQL 编译

在一个数据库中执行任何 SQL 语句之前，必须首先准备 SQL 语句。在此过程中，SQL 语句会被简化为经过优化器查询重写后的优化的 SQL 语句，DB2 优化器随后可对此语句进行分析。这条优化后的 SQL 语句就是所谓的 SQL 语句的存取计划(access plan)，在整个优化过程中发挥作用。图 12-1 展示了 SQL 语句的编译过程。

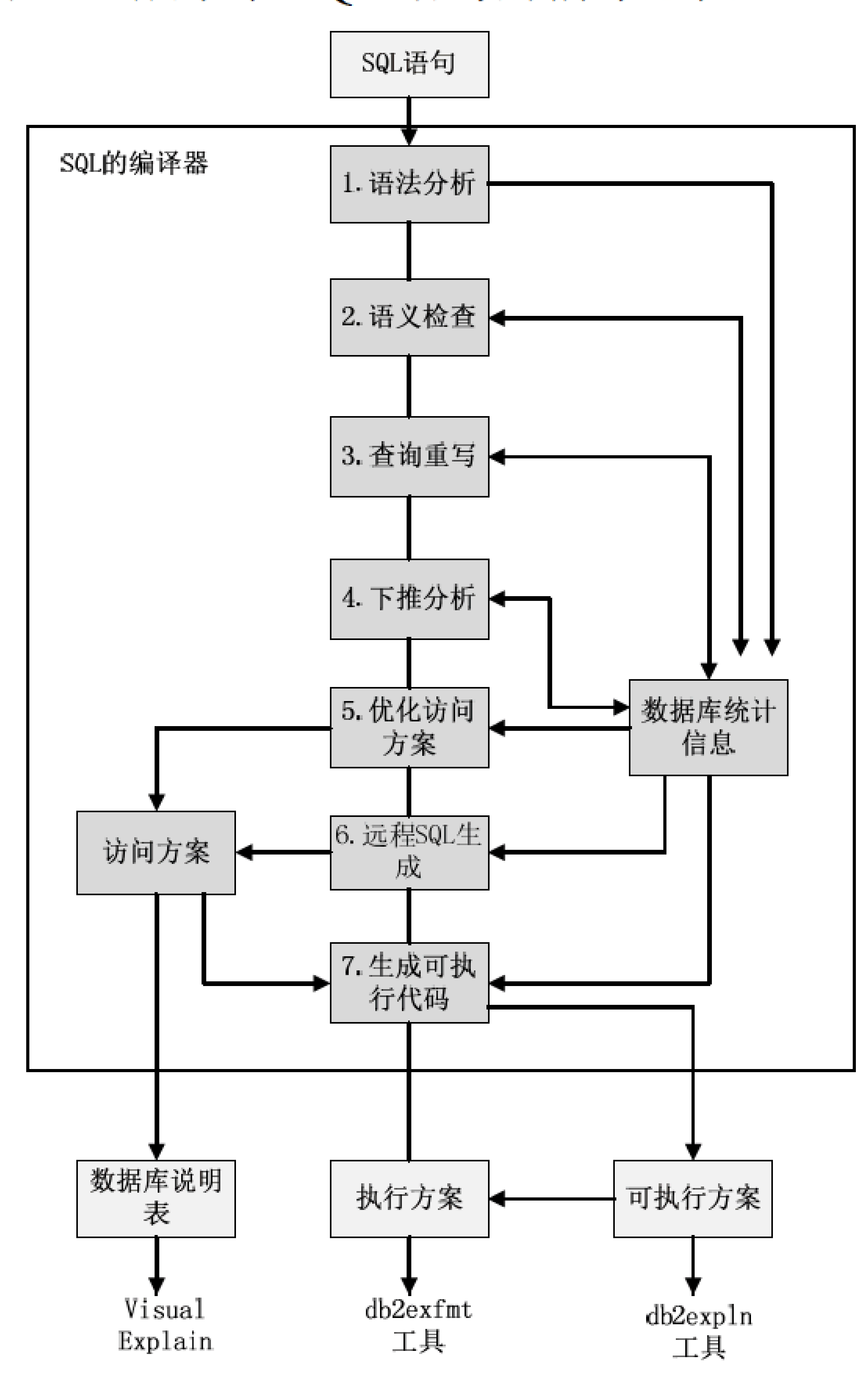


图 12-1　SQL 编译过程

SQL 编译过程的最终输出是一个访问计划。访问计划是 DB2 用于执行 SQL 语句的路径和步骤。这是由所有可用解释工具显示的信息。乍看上去，访问计划似乎非常复杂。但在具备了一定经验之后，您会很快发现它们实际上非常容易阅读和分析。

3. 激活 Visual Explain

只要收集到了全面解释(explain all)和(或)解释快照(explain snapshot)数据，关于数据收集实现和方法的信息就会记录在 EXPLAIN_INSTANCE 解释表中。您可随时通过“说明语句历史(Explained Statement History)记录窗口查看此信息。要激活该窗口，请在 Control Center 中高亮显示适当的数据库，并在 Control Center 菜单中选择 **Selected > Show Explained Statement History** 即可。图 12-2 展示了在为一条 SQL 语句收集了解释快照数据之后，“说明语句历史(Explained Statement History)记录窗口的外观。

提示：

本例中使用的 DB2 环境是中文环境，如果读者数据库环境是英文，只需对照相应术语的中英文翻译即可。



图 12-2 “说明语句历史(Explained Statement History)记录”窗口

一旦打开了“说明语句历史记录”窗口，您就可以使用 Visual Explain 来分析解释的快照数据了，显示这些数据的方法是，高亮显示一条记录，并在窗口的主菜单中选择 **Statement (语句)> Show Access Plan(显示访问计划)**。图 12-3 展示了以这种方法为以下查询创建的 Access Plan Graph 窗口(此查询可在随 DB2 提供的 SAMPLE 数据库上执行)：

```
select name,balance from account where acct_id=47030
```

另一方面，还可为新查询收集解释快照数据，相应的访问计划可通过在“说明语句历史记录”窗口的主菜单中选择 **Statement > Explain Query...**显示出来。在选中这些菜单项时，Explain Query Statement 窗口将打开，并提示您为查询输入文本。图 12-4 展示了以一个简单的查询填充后的 Explain Query Statement 窗口。

Access Plan Graph 窗口中展示的访问计划的每个组件都可供单击，单击后即可看到关于该组件的详细信息。例如，若选中图 12-3 所示的访问计划中的操作符，则 Operator Details 窗口中将显示如图 12-5 所示的详细信息。

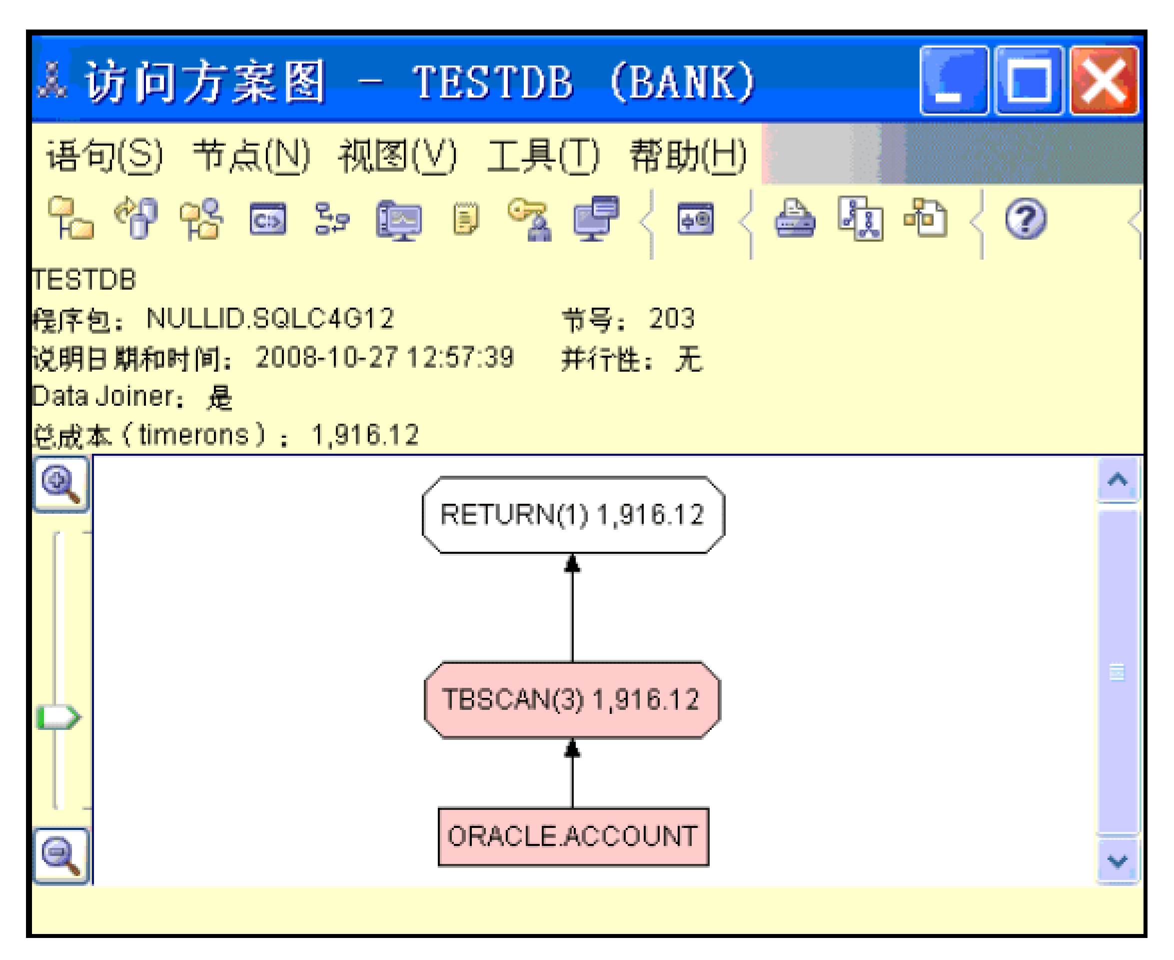


图 12-3 Access Plan Graph 窗口

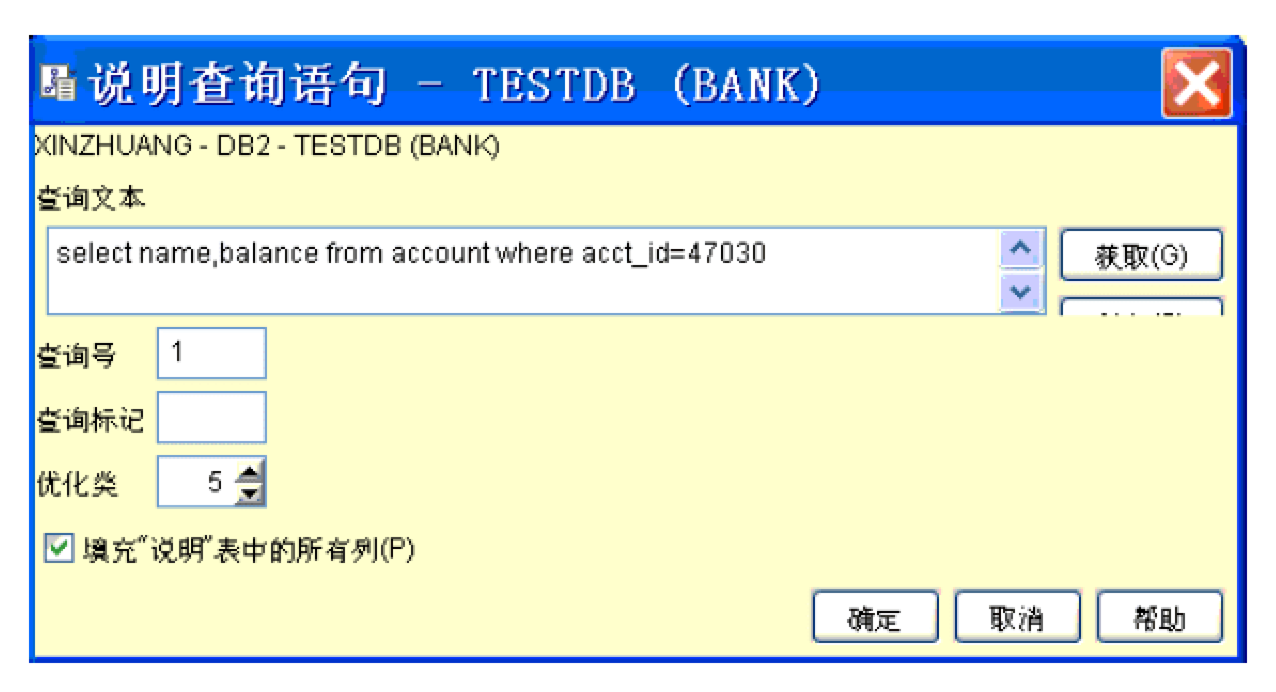


图 12-4 Explain Query Statement 窗口



图 12-5 Operator Details 窗口

在分析一个访问计划以定位性能瓶颈时，最好尝试单击所有不同的对象类型，以便充分了解已有的查询信息。

4. Visual Explain 组件

您可能已经注意到，Access Plan Graph 窗口中提供的输出(参见图 12-3)由层次化图形构成，表示处理为指定查询选定的访问计划时所必需的不同组件。计划中的各组件都显示为一种称为节点的图形对象。可存在两种类型的节点：

- **操作符(Operator)**。操作符节点用于确定是否必须在数据上执行一项活动，或者通过表或索引生成的输出。
- **操作对象(Operand)**。操作对象节点用于确定对其进行操作的实体(例如，表可以是一个表扫描操作符的操作对象)。

操作对象

典型情况下，操作对象节点用于确定表、索引和表队列(表队列用于使用了内部分区并行操作的情况)，它们在层次图中的符号分别是矩形(表)、菱形(索引)和平行四边形(表队列)。图 12-6 给出了表和索引操作对象的示例。

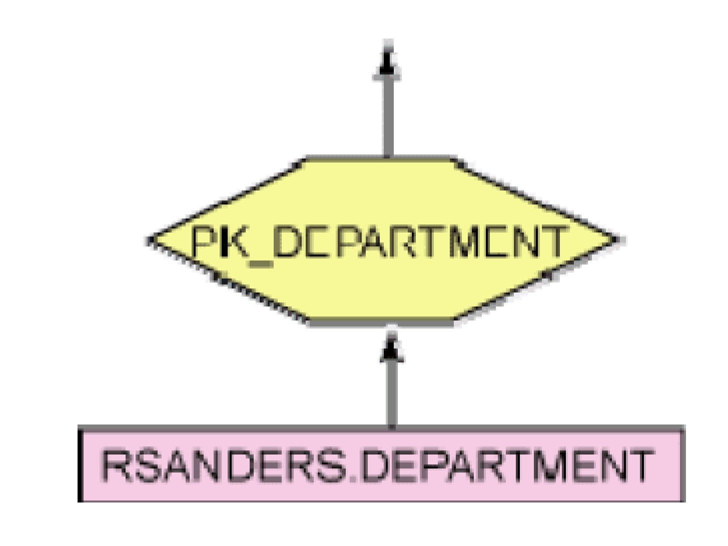


图 12-6　表和索引操作对象示例

操作符

另一方面，操作符节点用于确定从插入操作到索引或表扫描的一切活动。操作符节点在层次图中的符号为八边形，表示数据的访问方法、表的连接方法，以及其他一些因素，例如是否要执行排序操作等。表 12-1 列出了访问计划层次图中较为常见的操作符。

表 12-1　常见 Visual Explain 操作符

操　作　符	所执行的操作
CMPEXP	计算表达式(仅用于调试模式)
DELETE	从表中删除行
EISCAN	扫描用户定义的索引，产生一系列简化的行

(续表)

操　作　符	所执行的操作
FETCH	使用指定的记录标识符从表中获取列
FILTER	通过应用一个或多个谓词过滤数据
GENROW	生成一个行表
GRPBY	按指定列或函数的公共值组织行，并对集合函数求值
HSJOIN	显示一个散列连接，其中一个或多个表在连接列上是混编的
INSERT	向表中插入行
IXAND	对两个或多个索引扫描所得到的行标识符(RID)进行 AND 运算
IXSCAN	使用可选的启动/停止条件扫描表索引，产生有序的行流
MSJOIN	显示合并连接，其中外部和内部表必须按连接谓词的顺序排列
NLJOIN	显示嵌套循环连接，为外部表中的各行访问内部表一次
PIPE	翻译行(仅用于调试模式)
RETURN	将查询返回的数据显示给用户
RIDSCN	扫描一个行标识符(RID)列表，该列表是从一个或多个索引中获得的
RPD	远程计划的操作符。与 DB2 V8 中的 SHIP 操作符极为类似(之前版本中的 RQUERY 操作符)，唯一的不同在于它不包含 SQL 或 XQuery 语句
SHIP	从远程数据源中检索数据。在联合系统中使用
SORT	按特定类的顺序排序行，可以选择消除重复条目
TBSCAN	通过直接从数据页中读取所有数据而检索行
TEMP	将数据存储在临时表中以便读回(很可能要读回多次)
TQUEUE	在数据库代理之间传输表数据
UNION	串联来自多个表的行流
UNIQUE	消除特定列值重复的行
UPDATE	更新表中的行
XISCAN	扫描 XML 表的索引
XSCAN	在一个 XML 文档节点子树中导航
XANDOR	允许为多个 XML 索引应用 ANDed 和 ORed 谓词

图 12-7 中展示了一些更为常见的操作符的示例。在这个示例中，执行了 3 种不同的行动：两个表执行了表扫描，两个数据集使用散列连接算法连接。

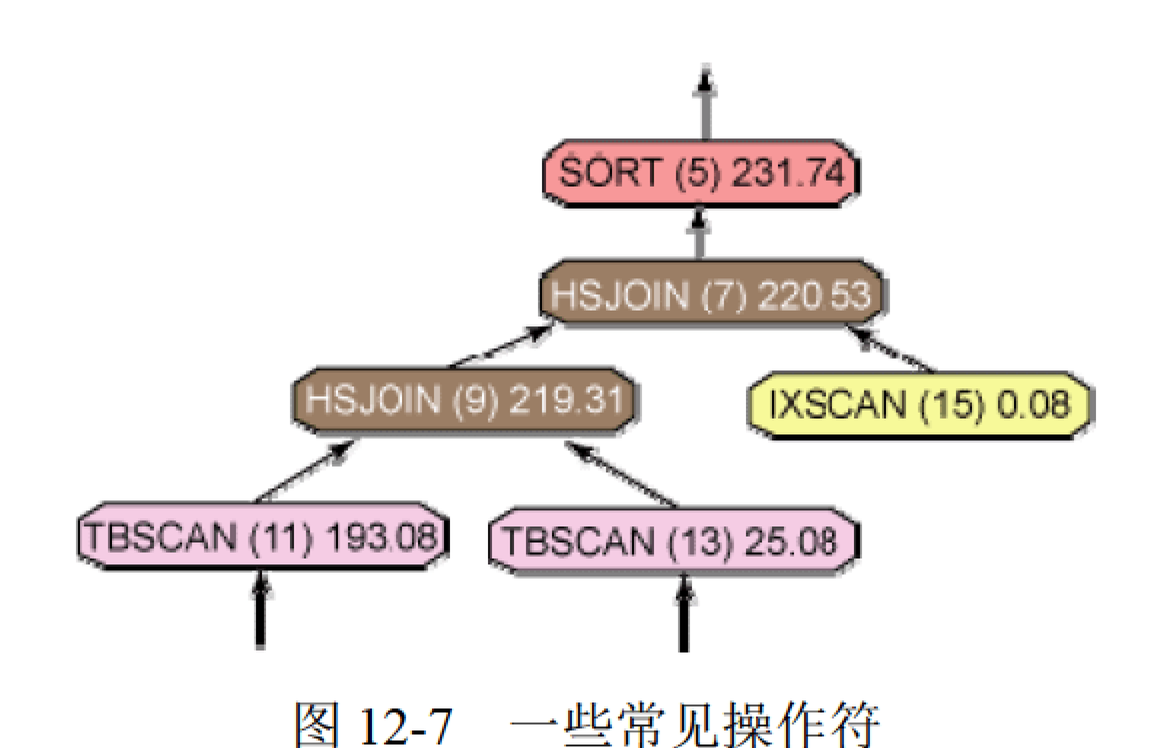


图 12-7 一些常见操作符

连接符和 RETRUN 操作符

箭头说明数据从一个节点流向另一个节点的方式，它将层次图中的所有节点连接在一起，RETURN 操作符通常用于终止这一过程。RETURN 操作符表示最终结果集产生，并包含关于查询的汇总信息以及所完成的 SQL 语句返回的内容。使用 RETURN 显示的 timeron 值表示按 timeron 度量的时间总长度，是完成查询所必需的时间。图 12-8 展示了 RETURN 操作符的一个示例。

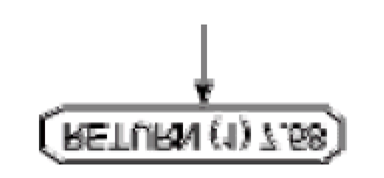


图 12-8 RETURN 操作符

5. 影响查询性能的因素

数据库环境的配置参数、统计信息、I/O 设计和用于准备查询的查询优化级别对于查询的准备方式、执行方式有着重大的影响。

显示优化参数

Visual Explain 可迅速汇总影响查询编译的所有参数，并在一个汇总窗口中显示出来。这个窗口就称为 Optimization Parameters 窗口，通过在 Access Plan Graph 窗口的主菜单中选择 **Statement > Show Optimization Parameters** 可调用此窗口。图 12-9 展示了 Optimization Parameters 窗口在激活时的外观。

Optimization Parameters 窗口中包含的部分配置参数：

- **AVG_APPLS**(平均应用程序)：此参数表示为数据库并发运行的应用程序平均数量。DB2 使用此信息来确定排序空间和缓冲池使用得有多么频繁，并确定查询能够使用的空间有多少。



图 12-9　“优化参数(Optimization Parameters)”窗口

- **SORTHEAP**(排序堆)：排序堆是执行排序时可用的内存空间数量。若排序需要的内存多于排序堆中可用内存，则部分排序数据将不得不分页到磁盘上(这会对性能造成严重的负面影响)。
- **LOCKLIST**(锁列表)：该参数表示 DB2 可用于存储各应用程序的锁定信息的内存数量。若锁列表空间过小，则 DB2 可能必须逐步升级(escalate)部分锁，以便为应用程序具有的所有锁腾出空间。
- **MAXLOCKS**(最大锁列表百分比)：该参数控制整个锁列表空间中有百分之多少的空间可为一个应用程序所有。若一个应用程序具有过多的锁，从而试图占用过多的内存，DB2 则升级部分锁，以释放锁列表中的空间。
- **NUM_FREQVALUES**(频繁值数)：DB2 RUNSTATS 实用工具使用频率值数来控制 DB2 将在内存中保留多少使用频率最高的值。优化器使用该信息来确定 WHERE 子句中的一个谓词将消耗结果集的多少百分比。
- **NUM_QUANTILES**(数据分位数)：DB2 RUNSTATS 实用工具使用分位数来控制为列数据捕获多少分位。增加分位数将给予 DB2 关于数据库中数据分布情况的更多信息。
- **DBHEAP**(数据库堆)：数据库堆控制数据库对象信息的可用内存量。对象包括索引、缓冲池和表空间。事件监控器和日志缓冲区信息也存储在这里。

- **CPUSPEED**(CPU 速度)：计算机的 CPU 速度。若此值设置为－1，则 DB2 使用 CPU 速度度量程序来确定恰当的设置。
- **BUFFPAGE 和缓冲池大小**：优化器可在优化数据过程中使用的缓冲池大小。增加或减少缓冲池大小会对访问计划产生显著影响。

12.1.2 db2expln

在包含嵌入式 SQL 语句的源代码文件绑定到数据库时(无论是作为预编译流程的一部分，还是在延迟绑定过程中)，DB2 优化器将分析遇到的每一条静态 SQL 语句，并生成一个相应的访问计划，此访问计划随后以程序包的形式存储在数据库中(syscat.packages)。给定数据库名称、包名称、包创建者 ID、部分号(若指定了部分号为 0，则处理包的所有部分)，db2expln 工具即可为存储在数据库系统目录中的任何包解释并说明其访问计划。由于 db2expln 工具直接处理包而非全面解释数据或解释快照数据，因而通常用来获取那些已选定用于未捕获其解释数据的包的访问计划的相关信息。但由于 db2expln 工具仅可访问已存储在包中的信息，因而只能说明所选的最终访问计划的实现，不能提供特定 SQL 语句优化方式的信息。

若使用额外的输入参数，db2expln 工具则还可用于解释动态 SQL(不包含参数标记的动态 SQL 语句)语句(db2expln 的早期版本不支持动态 SQL(如 DB2 V7)，需要使用 dynexpln 工具，有了新版本后，可以不再使用 dynexpln，只用 db2expln 就可以了)。

查看静态 package 例子：

对于数据库 SAMPLE 中的程序包 DB2INST1.P0203450，使用下面命令：

```
db2expln -d SAMPLE -g -c db2inst1 -p P0203450 -s 0 -t
```

其中：

- -g 是指给出存取计划的图形输出(用字符模拟的)
- -s 0 是指分析所有的 SQL 命令

下面是相应的输出：

```
DB2 Universal Database Version 8.2, 5622-044(c) Copyright IBM Corp. 1991, 2007
Licensed Material - Program Property of IBM
IBM DATABASE 2SQLExplain Tool
Processing package DB2INST1.P0203450.
******************** PACKAGE *********************
Package Name = DB2INST1.P0203450
--------Prep Date = 2007/12/17
--------Prep Time = 16:02:04
--------Bind Timestamp = 2007-12-17-16.02.04.373971
```

```
--------Isolation Level ---------= Cursor Stability
--------Blocking--------------- = Block Unambiguous Cursors
--------Query Optimization Class = 5
--------Partition Parallel -------= No
--------Intra-Partition Parallel = No
--------Function Path----------- = "SYSIBM", "SYSFUN", "DB2INST1"
Processing Section 1.
-------------------- SECTION ----------------------
-------------------- SECTION ----------------------
Section = 2
SQL Statement:
update STAFF set NAME = 'test'  where ID = 350
Estimated Cost = 75
Estimated Cardinality = 2
Access Table Name = DB2INST1.STAFF ID = 2,3
| #Columns = 2
| Relation Scan-------------------全表扫描
| | Prefetch: Eligible
| Lock Intents
| | Table: Intent Exclusive
| | Row : Update
| Sargable Predicate(s)
| | #Predicates = 1
Update: Table Name = DB2INST1.STAFF ID = 2,3
End of section
优化器 Plan:
------UPDATE
------( -2)
------/-- \
-TBSCAN --Table: 全表扫描
--( 3) --DB2INST1
---| -----STAFF
Table:
DB2INST1
STAFF
......
```

可以用下面命令找出数据库中存在的程序包：

```
db2 "select pkgschema, pkgname from syscat.packages where pkgname =' P0203450'"
```

上面例子中的程序包 DB2INST1.P0203450 是一个存储过程。

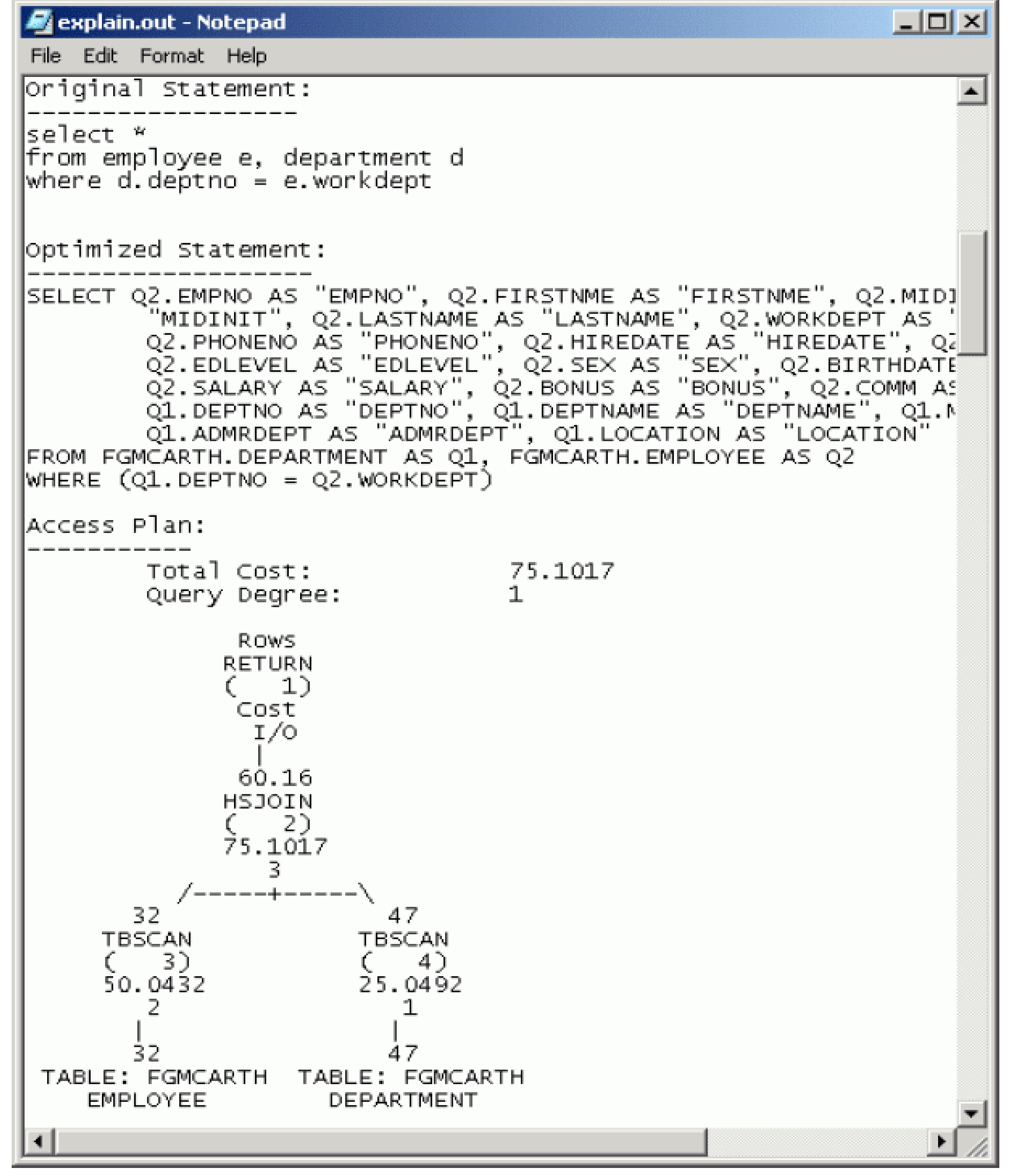
查看动态 SQL 的例子：

```
C:\>db2expln -d sample -q "select * from employee" -t
DB2 Universal Database Version 9.5, 5622-044(c) Copyright I
Licensed Material - Program Property of IBM
IBMDB2Universal DatabaseSQLand XQUERY Explain Tool
DB2 Universal Database Version 9.5, 5622-044(c) Copyright I
Licensed Material - Program Property of IBM
IBMDB2Universal DatabaseSQLand XQUERY Explain Tool
******************** DYNAMIC ******************************
==================== STATEMENT =============================
        Isolation Level          = Cursor Stability
        Blocking                = Block Unambiguous Cursors
        Query Optimization Class = 5
        Partition Parallel       = No
        Intra-Partition Parallel = No
       SQLPath                = "SYSIBM", "SYSFUN", "SYSP
                                 "ORACLE"
Statement:
  select *
  from employee
Section Code Page = 1208
Estimated Cost = 7.612985
Estimated Cardinality = 42.000000
Access Table Name = ORACLE.EMPLOYEE  ID = 2,6
|  #Columns = 14
|  Avoid Locking Committed Data
|  Relation Scan
|  |  Prefetch: Eligible
|  Lock Intents
|  |  Table: Intent Share
|  |  Row  : Next Key Share
|  Sargable Predicate(s)
|  |  Return Data to Application
|  |  |  #Columns = 14
Return Data Completion
End of section
```

12.1.3　db2exfmt

与 db2expln 工具不同，db2exfmt 工具用于直接处理已收集并存储在解释表中的全面解释数据或解释快照数据。给定数据库名和其他限定信息，db2exfmt 工具将在解释表中查询信息、格式化结果，并生成一份基于文本的报告，此报告可直接显示在终端上或写入 ASCII 文件。

所有 explain 输出(包括 Visual Explain)都是从下往上读的，如图 12-10 所示。




```
Original Statement:
------------------
select *
from employee e, department d
where d.deptno = e.workdept


Optimized Statement:
-------------------
SELECT Q2.EMPNO AS "EMPNO", Q2.FIRSTNME AS "FIRSTNME", Q2.MIDI
        "MIDINIT", Q2.LASTNAME AS "LASTNAME", Q2.WORKDEPT AS '
        Q2.PHONENO AS "PHONENO", Q2.HIREDATE AS "HIREDATE", Q2
        Q2.EDLEVEL AS "EDLEVEL", Q2.SEX AS "SEX", Q2.BIRTHDATE
        Q2.SALARY AS "SALARY", Q2.BONUS AS "BONUS", Q2.COMM AS
        Q1.DEPTNO AS "DEPTNO", Q1.DEPTNAME AS "DEPTNAME", Q1.M
        Q1.ADMRDEPT AS "ADMRDEPT", Q1.LOCATION AS "LOCATION"
FROM FGMCARTH.DEPARTMENT AS Q1, FGMCARTH.EMPLOYEE AS Q2
WHERE (Q1.DEPTNO = Q2.WORKDEPT)

Access Plan:
-----------
        Total Cost:             75.1017
        Query Degree:           1

              Rows
             RETURN
             (   1)
              Cost
               I/O
               |
              60.16
             HSJOIN
             (   2)
             75.1017
                3
          /-----+-----\
      32                47
    TBSCAN            TBSCAN
    (   3)            (   4)
    50.0432           25.0492
       2                 1
      |                 |
      32                47
 TABLE: FGMCARTH   TABLE: FGMCARTH
    EMPLOYEE          DEPARTMENT
```

图 12-10 explain 输出

这里不像 Visual Explain 那样将所有细节显示在不止一个屏幕上，而是将所有细节列在一个输出文件中。图 12-10 中每个操作符都编了号，当您往下查看时，每个操作符都将被详细解释。例如，图中的一个操作符可以作如下解释：

```
        5.7904 - # of rows returned(based on statistics calculation)
        HSJOIN  - type of operator
       (   2)- operator #
        75.1017     - cumulative timerons
          3     -I/Ocosts
```

返回的行数、timeron(cost)数和 I/O 都是优化器估计的，在某些情况下可能与实际数字不符。timeron 的概念我们前面已经讲过，它是 DB2 的成本度量单元，用于给出对数据库服务器在执行同一查询的两种计划时所需的资源或成本的粗略估计。估计时计算的资源包括处理器和 I/O 的加权成本。

您可以使用 db2exfmt 来解释单独一条语句。例如：

```
    explain all for
          SQL_statement
```

```
db2exfmt -d
        dbname -g tic -e
        explaintableschema -n % -s % -w -1 -# 0 -o
        outfile
```

如果为用";"隔开的几条"explain all'语句构建一个文本文件，就可以一次解释多条语句：

```
db2 -tf
        file_with_statements
db2exfmt -d
        dbname -g tic -e
        explaintableschema -n % -s % -w % -# 0 -o
        outfile
```

12.1.4　各种解释工具比较

如您所见，可用于显示全面解释数据和解释快照数据的不同工具有着很大的差异，无论是在复杂性方面还是在功能方面。表 12-3 总结了几种可用工具，并强调了各自的特征。要使解释工具发挥出最好的效果，您应在选择工具时考虑您的环境和需求。

表 12-3　可用解释工具的比较

所 需 特 征	Visual Explain	db2exfmt	db2expln
用户界面	图形化	基于文本	基于文本
快速但粗略的静态 SQL 分析	否	否	是
静态 SQL 支持	是	是	是
动态 SQL 支持	是	是	是
CLI 应用程序支持	是	是	否
详细的 DB2 优化器信息可用	是	是	否
适于分析多条 SQL 语句	否	是	是

12.1.5　如何从解释信息中获取有价值的建议

当分析 explain 的输出信息时，应该注意以下问题：

- 对相同的一组列和基本表使用的 ORDER BY、GROUP BY 或 DISTINCT 操作符将从索引或物化查询表(MQT)中受益，因为消除了排序。explain 用来帮助确保索引被正确地用于连接谓词、本地谓词以及 GROUP BY 和 ORDER BY 子句，以避免排序。
- 代价较高的操作，例如大型排序、排序溢出以及对表的大量使用，都可以受益于更多的排序空间(sortheap)、更好的索引、更新的统计信息或调优的 SQL。

- 表扫描(relation scan)也可以从索引中受益。
- 完全索引扫描或无选择性的索引扫描，其中不使用 start 和 stop 关键字，或者使用这两个关键字，但是有一个很宽的取值范围。这样的扫描性能会很差，尝试调整。
- 表的连接类型，利用您对表中数据的了解来确定正采用的连接类型是否正确，以及正在用于连接的内表和外表的表是否正确。
- 未充分地利用索引。查询是否按您的希望使用了索引，应确保未在您理所当然地认为应该具有索引的表上进行表扫描。此问题可通过查看执行计划轻松应对。若确实存在索引，则检查基数或索引键的顺序。确保索引的集群度。
- 表基数和'SELECT *'的使用。有时，由于您要返回的列数，DB2 优化器会判定扫描整个表的速度更快。有可能表非常小，也有可能扫描索引并返回大量行(即返回表中的所有列)的效率很低。尝试仅返回那些您确实需要的列。查看查询各部分返回的列，观察您是否确实需要这些列，并观察这是否是表扫描发生的原因。同样，考虑使用索引中包含的列。
- 优化级别设置是否合适。

12.2 索引设计工具(db2advis)

12.2.1 DB2 Design Advisor(db2advis)

DB2 UDB V8.2 引入了一种名为 Design Advisor 的新工具，它对 DB2 早期版本中的 db2advis 工具做了更多扩展，提供了更强大的功能，该工具使用范围更广，可用来替代 db2advis，如图 12-11 所示。

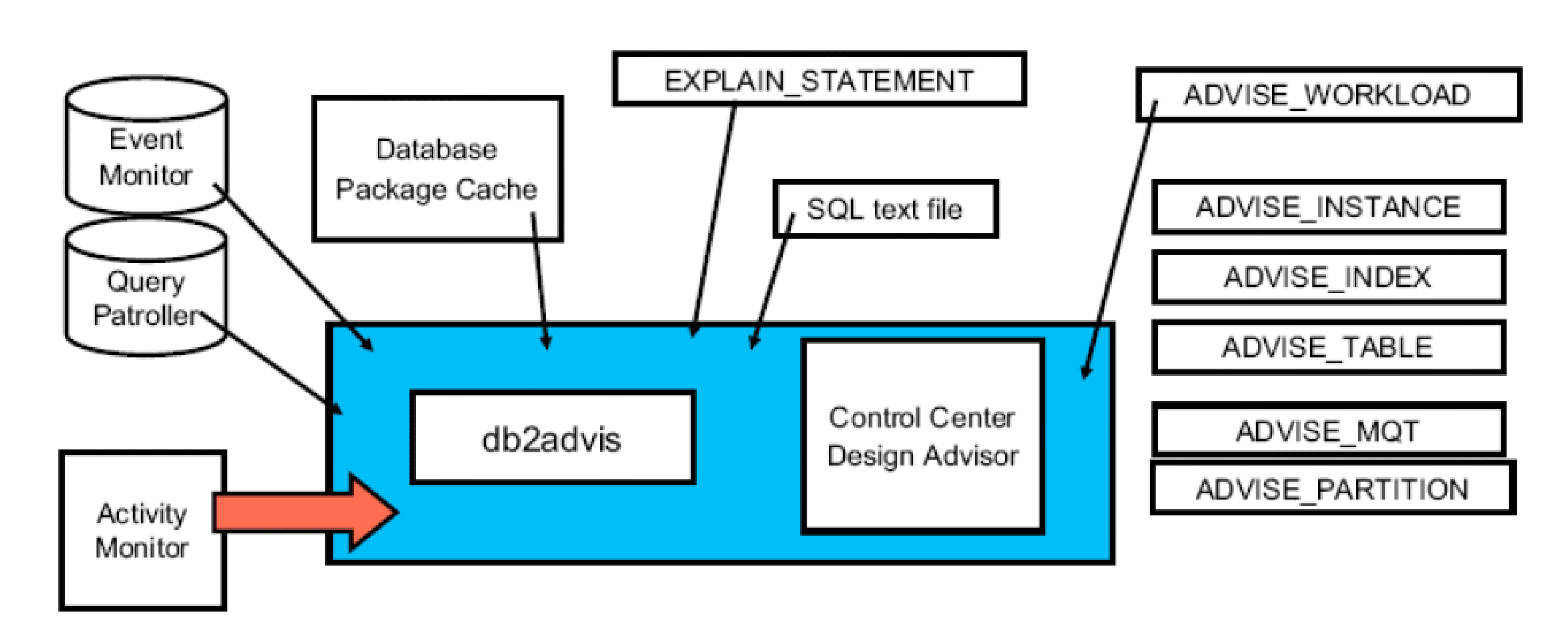


图 12-11　Design Advisor

Design Advisor 所提供的建议能与数据库调优专家的建议相媲美。对于非专家来说，该工具的好处是可以获得更好的设计。对于专家来说，Design Advisor 可以节省他们宝贵的时间，因为 Design Advisor 可以提供一个初始的设计，然后由专家进一步改进设计。Design Advisor 还可以提供对专家的设计的独立确认。

要使用 Design Advisor，首先需要在相应数据库里执行 EXPLAIN.DDL 存储 Advisor 各种数据。

```
cd $INSTHOME/sqllib/misc
db2 -tf EXPLAIN.DDL
```

使用 Design Advisor 的第一步是收集和描述提供给 Design Advisor 的工作负载。可以让 Design Advisor 从以下输入获取工作负载：

- 最近的 SQL 语句(来自动态 SQL 快照)
- Query Patroller 语句(更适用于数据仓库数据库)
- 静态 SQL 语句(来自应用程序包)
- 解释后的 SQL 语句
- Event Monitor 语句
- 来自一个包含事务的文件
- Activity Monitor 捕获的 SQL 语句

我们可以为每个事务(即每个 SQL)赋予 1、10、100 或 1000 的频率。这将导致 Design Advisor 对频率值为 10 的事务的重视程度是对频率值为 1 的事务的重视程度的 10 倍。

收集完事务并设置好每个事务的频率后，我们就可以运行命令 db2advis 获取关于索引的建议，如下所示。如果创建了索引，SQL 执行成本可以提高 32.42%。

```
db2advis -d sample -i top.sql -disklimit 2 -o newindex.ddl > advis.out
cat advis.out
execution started at timestamp 2005-12-17-11.54.08.236000
found [3]SQLstatements from the input file
Recommending indexes...
total disk space needed for initial set [ 0.817] MB
total disk space constrained to [ 2.000] MB
Trying variations of the solution set.
Optimization finished.
1 indexes in current solution
[1118.4197] timerons(without recommendations)
[755.8014] timerons(with current solution)
[32.42%] improvement
-- LIST OF RECOMMENDED INDEXES
-- ===========================
```

```
    -- index[1], 0.817MB
    CREATE INDEX "ADMIN"."IDX411171655470000" ON "ADMIN"."HISTORY" ("TELLER_ID"
ASC, "BRANCH_ID" ASC) ALLOW REVERSE SCANS ;
    COMMIT WORK ;
    RUNSTATS ON TABLE "ADMIN"."HISTORY" FOR INDEX "ADMIN"."IDX411171655470000" ;
    COMMIT WORK ;
```

12.2.2 DB2 Design Advisor(db2advis)案例讲解

现在我们将讲解一个案例，不过这次使用的是命令行，而不是 GUI。

下面是这个案例中使用到的命令：

```
    db2advis -d sample -i top1.sql -m IMCP -k LOW  -l 700 -c TEMP_TBS -f
```

要点包括：

-m IMCP：规定 Design Advisor 应该考虑新的索引(I)、MQT(M)，将标准的表转换成 MDC 表(C)，并且重新为已有的表分区(P)。默认情况是只考虑索引。

-k LOW：规定将工作负载压缩到 low 级别。结果，Design Advisor 将分析您提供的更大一组的工作负载。默认情况是中等(medium)。

-l 700：规定任何新的索引、MQT 等都不能消耗多于 700MB 的空间。默认情况是数据库总体规模的 20%。

-c TEMP_TBS：规定使用表空间 TEMP_TBS 作为生成 MQT 建议的临时工作空间。如果您想要 MQT 建议，并且正在运行一个 DPF(多分区)示例，那么这个选项是必需的。否则，这个参数是可选的。

另一个有用的选项(没有给出)是-o *output_file*。该选项保存脚本，以便在一个文件中创建建议的对象。

当该命令执行时，它描述正在进行的工作，下面显示了其中的一部分。至此，Design Advisor 已经生成了除 MDC 之外的所有有关对象的建议。

```
    Cost of workload with all recommendations included [1306186] timerons
    27 indexes in current solution
    3 partitionings in current solution
    8 MQTs in current solution
```

建议集由 27 个索引(新索引或已有的索引)、3 个分区(即与 DPF 相关的更改，例如新的分区键或表空间)，以及 8 个 MQT(新的或已有的)。

接下来，Design Advisor 分析 MDC，并在完成时显示以下信息：

```
    3 clustering dimensions in current solution
```

```
[12305400] timerons(without any recommendations)
[1042873] timerons(with current solution)
[91.53%] improvement
```

“3 clustering dimensions”意味着 Design Advisor 建议 3 个 MDC 维。这 3 个维可以同时在一个表上，也可以在不同的表上，例如，3 个维都在表 A 上，或者其中 1 个维在表 A 上，另外两个维在表 B 上。性能统计信息指的是所有建议的性能，而不仅仅是 MDC 建议的性能。“timerons(without any recommendations)”这一项指的是现有设计能取得的性能，而“with current solution”指的是实施这些建议后估计能取得的性能。

接着，Design Advisor 以 DDL 格式显示建议，并且该 DDL 已经被注释掉。这些建议以如下顺序出现：

- 包括 MDC 或分区建议的基本表
- MQT 建议(首先是新的 MQT，然后是要保留的已有的 MQT，最后是未使用的 MQT)
- 新的集群索引(如果有的话)
- 索引建议(新的，保留的，然后是未使用的)

关于更改一个表的建议如下所示：

```
-- CREATE TABLE "ORACLE"."LINEITEM"("L ORDERKEY BIGINT NOT NULL,
-- "L_PART" INTEGER NOT NULL,
-- "L SUPPKEY" INTEGER NOT NULL,
-- "L_LINENUMBER" INTEGER NOT NULL,
-- "L SHIPINSTRUCT" CHAR(25)  NOT NULL,
(11 other columns omitted from this example)
-- MDC409022109290000 GENERATED ALWAYS AS(((INT(L SHIPDATE))/7) )
-- ---- PARTITIONING KEY("L_PARTKEY") USING HASHING
-- ---- IN "TPCDLADT"
-- ORGANIZE BY(
-- MDC409022109290000,
-- L_SHIPINSTRUCT )
-- PARTITIONING KEY(L ORDERKEY) USING HASHING
-- IN TPCDLDAT
--;
-- COMMIT WORK ;
```

注意，这里建议了一个新的分区键(L_ORDERKEY)，用以替代当前的分区键(L_PARTKEY)，后者被注释掉了。这个表的 MDC 建议(ORGANIZE BY 子句)包括两个维：一个生成的列(INT(L_SHIPDATE/7))和一个已有的列(L_SHIPINSTRUCT)。

输出中接下来的是关于 MQT 的建议，如下所示：

```
-- LIST OF RECOMMENDED MQTs
-- ========================
-- MQT MQT409022041400000 can be created as a refresh immediate MQT
-- mqt[1],   0.009MB
CREATE SUMMARY TABLE "ADVDEMO2"." MQT409022041400000"
   AS (SELECT Q6.C0 AS "C0", Q6.C1 AS "C1", …additional details omitted here…)
   DATA INITIALLY DEFERRED REFRESH IMMEDIATE PARTITIONING KEY (C8)
   USING HASHING IN TPCDLDAT ;
COMMIT WORK;
REFRESH TABLE "ADVDEMO2"." MQT409022041400000";
COMMIT WORK;
RUNSTATS ON TABLE "ADVDEMO2"." MQT409022041400000";
COMMIT WORK;
-- MQT MQT4090220415300000 can be created as a refresh immediate MQT
(… DDL to create this table follows…)
```

MQT 建议包括：估计的大小、使用的表空间、分区键(如果适用的话)、刷新类型(立即或延迟)以及这个表是否是一个基本表的复制品(由 REPLICATE 关键字表明)，在本案例中不是。

最后，Design Advisor 以下面显示的信息结束：

```
8604 solutions were evaluated by the advisor
DB2 Workload Performance Advisor tool is finished.
```

对于经常使用的查询以及大型或复杂的查询，将这些 SQL 语句作为输入，使用 Design Advisor 来建议索引。

此外，针对由完整应用程序测试填充的动态 SQL 缓存重新运行 Design Advisor。这允许根据实际工作负载和 SQL 语句的执行频率建议索引。确保在 Design Advisor 执行之前运行了 RUNSTATS。

12.3 基准测试工具 db2batch

12.3.1 db2batch

基准测试是从各种不同方面(例如数据库响应时间、CPU 和内存使用情况)对应用程序进行评测的一个过程。基准测试基于一个可重复的环境，以便能够在相同的条件下运行相同的测试。之后，对测试收集到的结果可以进行评估和比较。

db2batch 是一种基准测试工具，它以一组 SQL 和/或 XQuery 语句作为输入，动态地准备语句和描述语句，并返回一个结果集。取决于 db2batch 命令中所使用的选项，结果集可

以返回这些语句的执行时间、关于内存使用情况(例如缓冲池)的数据库管理器快照和缓存信息。

可以在一个 flat 文件或标准输入中指定要运行基准测试的语句。在输入文件中可以设置很多控制选项。指定这些选项的语法是：--#SET *control_option value*。下面是包含控制选项的一个输入文件的例子：

```
-- db2batch.sql
-- ------------
--#SET PERF_DETAIL 3
--#SET ROWS_OUT 5
-- This query lists employees, the name of their department
-- and the number of activities to which they are assigned for
-- employees who are assigned to more than one activity less than
-- full-time.
--#COMMENT Query 1
select lastname, firstnme,  deptname, count(*) as num act
from employee, department, emp_act
where employee.workdept = department.deptno and
     employee.empno = emp_act.empno and     emp_act.emptime < 1
        group by lastname, firstnme, deptname having count(*) > 2;
--#SET PERF_DETAIL 1
--#SET ROWS OUT 5
--#COMMENT Query 2
select lastname, firstnme,   deptname, count(*) as num act
from employee, department, emp_act
where employee.workdept = department.deptno and
     employee.empno = emp_act.empno and      emp_act.emptime < 1
group by lastname, firstnme, deptname having count(*) <= 2;
```

- 选项 PERF_DETAIL 3 意味着将返回关于花费的时间和数据库管理器、数据库及应用程序的快照这些性能方面的细节。
- 选项 ROWS_OUT 5 意味着无论查询返回的实际行数是多少，只从结果集中取 5 行。
- COMMENT Query1 将语句命名为 Query1。

下面的命令在 SAMPLE 数据库上调用基准测试工具，输入文件为 db2batch.sql。

```
db2batch -d sample -f db2batch.sql
```

这个命令将返回查询的结果集(限 5 行)和查询所花费的时间及 CPU 时间。另外还返回数据库管理器、数据库和应用程序快照。由于输出很大，因此这里只显示 db2batch 命令的概要。

```
* Summary Table:
Type      Number      Repetitions Total Time(s) Min Time(s)   ...
--------- ----------- ----------- -------------- --------------
Statement         1           1       0.052655       0.052655 ...
Statement         2           1       0.004518       0.004518 ...
...Max Time(s)   Arithmetic Mean Geometric Mean Row(s) Fetched Row(s) Output
-------------- --------------- -------------- -------------- -------------
...      0.052655       0.052655       0.052655              5           5
...      0.004518       0.004518       0.004518              8           5

* Total Entries:              2
* Total Time:                0.057173 seconds
* Minimum Time:              0.004518 seconds
* Maximum Time:              0.052655 seconds
* Arithmetic Mean Time:       0.028587 seconds
* Geometric Mean Time:        0.015424 seconds
```

db2batch 命令支持很多选项。这里只列出其中一些选项，让您对这个工具的功能有所了解。

- -m *parameter_file* 用参数值指定用于绑定到 SQL 语句参数占位符的一个输入文件。
- -r *result_file* 指定存放命令结果的输出文件。
- -i *short*|*long*|*complete* 指定从哪个方面测量所花费的时间。*short* 测量运行每条语句所花费的时间。*long* 测量运行每条语句所花费的时间，包括语句之间的开销。*complete* 测量运行每条语句所花费的时间，分别报告准备、执行和取数据的时间。
- -iso 指定语句使用的隔离级别。默认情况下，db2batch 使用 Repeatable Read 隔离级别。

12.3.2　db2batch 基准程序测试分析示例

基准程序的输出应该包括每个测试的标识、程序执行的迭代、语句号和执行的计时。在经过一系列测量之后，基准程序测试结果的摘要可能类似于如下所示：

```
Test     Iter.   Stmt    Timing        SQLStatement
Numbr    Numbr   Numbr  (hh:mm:ss.ss)
     002      05      01     00:00:01.34    CONNECT TO SAMPLE
     002      05      10     00:02:08.15    OPEN cursor_01
     002      05      15     00:00:00.24    FETCH cursor_01
     002      05      15     00:00:00.23    FETCH cursor_01
     002      05      15     00:00:00.28    FETCH cursor_01
     002      05      15     00:00:00.21    FETCH cursor_01
     002      05      15     00:00:00.20    FETCH cursor_01
```

```
        002      05      15     00:00:00.22    FETCH cursor_01
        002      05      15     00:00:00.22    FETCH cursor_01
        002      05      20     00:00:00.84    CLOSE cursor_01
        002      05      99     00:00:00.03    CONNECT RESET
```

注意：
上面报告中的数据仅供说明之用。它不表示测量的结果。

分析显示：CONNECT(语句 01)用去 1.34 秒，OPEN CURSOR(语句 10)用去 2 分钟 8.15 秒，FETCHES(语句 15)返回 7 行且延迟时间最长的为 0.28 秒，CLOSE CURSOR(语句 20)用去 0.84 秒，而 CONNECT RESET(语句 99)用去 0.03 秒。

如果您的程序可以定界 ASCII 格式输出数据，那么可以在以后将该数据导入数据库表或电子表格进行进一步统计分析。

基准程序报告的样本输出可能是：

```
           PARAMETER        VALUES FOR EACH BENCHMARK TEST
           TEST NUMBER       001      002      003     004      005
           locklist         63       63       63       63       63
           maxappls         8        8        8        8        8
           applheapsz        48       48       48       48       48
           dbheap           128      128      128      128      128
           sortheap          256      256      256      256      256
           maxlocks         22       22       22       22       22
           stmtheap         1024     1024     1024     1024     1024
          SQLSTMT         AVERAGE TIMINGS(seconds)
               01          01.34    01.34    01.35    01.35    01.36
               10          02.15    02.00    01.55    01.24    01.00
               15          00.22    00.22    00.22    00.22    00.22
               20          00.84    00.84    00.84    00.84    00.84
               99          00.03    00.03    00.03    00.03    00.03
```

注意：
上面报告中的数据仅供说明之用。它不表示任何测量的结果。

12.4 数据一致性检查工具

12.4.1 db2dart 及案例

可以使用 db2dart 命令来验证数据库及其对象的体系结构是否正确。还可以使用它来显示数据库控制文件的内容，以便从其他情况下可能无法访问的表中抽取数据。

要显示所有可能的选项，只需发出不带任何参数的 db2dart 命令。如果命令行中未显式指定一些需要参数的选项(如表空间标识)，那么会提示输入这些参数。

默认情况下，db2dart 实用程序将创建名为 databaseName.RPT 的报告文件。对于单一分区数据库分区环境，将在当前目录中创建该文件。对于多分区数据库分区环境，将在诊断目录的子目录中创建该文件。该子目录称为 DART####，其中####是数据库分区号。

db2dart 实用程序通过直接从磁盘中读取数据库中的数据和元数据来对其进行访问。因此，决不能对仍具有活动连接的数据库运行该工具。如果存在活动连接，那么该工具将不知道缓冲池中的页面或内存中的控制结构(此处是举例说明)，可能会报告假的错误结果。同样，如果对需要进行崩溃恢复或尚未完成前滚恢复的数据库运行 db2dart，那么由于磁盘上的数据性质不一致，可能会导致类似的不一致情况。

DBA 可以用 db2dart 对数据库作一致性检查，从而检查数据页和索引页是否有损坏，db2dart 是一个数据库级别的检查和修复工具。它可以检查表空间和表的完整性，以及检查数据页的页连续一致性。它类似 Sybase 数据库的 dbcc 工具。使用 db2dart 时，必须断开所有与数据库的连接，下面我们举一个例子：

```
    C:\>db2dart SAMPLE /db
    FYI: An active connection to the database has been detected.
        False errors may be reported.
        Deactivate all connections and re-run to verify.
    Warning: The database state is not consistent.
    Warning: Errors reported about reorg rows may be due to the inconsistent
state of the database.
                   DB2DART Processing completed with warning(s)!
                        Complete DB2DART report found in:
    C:\DOCUME~1\ALLUSE~1\APPLIC~1\IBM\DB2\DB2COPY1\DB2\DART0000\SAMPLE.RPT
```

我们可以查看 db2dart 生成的报告文件，对数据库的一致性作检查，判断数据库中是否有坏的数据页或索引页。关于 db2dart 工具还有很多高级内容。在此书中限于篇幅，我对 db2dart 和 inspect 工具不做过多的讲述。如果读者渴望了解更进一步的内容，那么请参见《深入解析 DB2》一书。

12.4.2 inspect 及案例

inspect 是 db2dart 的派生命令。inspect 是集成在 db2 引擎中的，因此，它可以利用 db2 的 bufferpool、数据预取等优势来获得命令和更好的执行性能。inspect 在操作的过程中访问数据库的对象，它在检查数据库对象过程中使用的隔离级别是 UNCOMMITTED READ。

inspect 会将检查出来的信息放在 dbm 的诊断数据的目录里面，如果在检查工作最后没

有发现错误，那么 inspect 就会自己将文件删除。如果发生错误，那么此文件将被保留。当然，这些 db2 的实用工具的输出都是未格式化的信息，需要进行格式化才能被阅读和分析，格式化的命令是 db2inspf。下面我们举个例子，在 Windows 平台上使用该命令检查表 emp_resume 的存储情况：

```
    1) db2 inspect check table name emp resume results keep res.ins
    (注：res.ins 文件将产生在 DB2 诊断日志所在路径下，如 C:\Documents and Settings\All
Users\Application Data\IBM\DB2\DB2COPY1\DB2 目录下)。
    2) db2inspf  res.ins res.txt  ----(格式化 INSPECT 的输出文件)，查看 resume.txt
文件，发现其中有类似以下输出：
    ATABASE: SAMPLE
    VERSION : SQL09050
    2008-11-20-14.33.57.093000
    操作：CHECK TABLE
    模式名：ORACLE
    表名：EMP_RESUME
    表空间标识：2  对象标识：8
    结果文件名：res.txt
         表阶段开始(有符号的标识：8，无符号的：8；表空间标识：2)：ORACLE.EMP_RESUME
         数据阶段开始。对象：8 表空间：2
         此表的索引类型为 2。
          遍历 DAT 扩展数据块映像，锚点为 672。
          已完成遍历扩展数据块映像。
          DAT 对象总结：总页数为 1 — 已使用的页数为 1 — 可用空间为 76 %
         数据阶段结束。
          索引阶段开始。对象：8 表空间：2
          遍历 INX 扩展数据块映像，锚点为 864。
          已完成遍历扩展数据块映像。
          INX 对象总结：总页数为 4 — 已使用的页数为 4
         索引阶段结束。
         LOB 阶段开始。对象：8 表空间：2
          遍历 LOB 扩展数据块映像，锚点为 736。
          已完成遍历扩展数据块映像。
          遍历 LBA 扩展数据块映像，锚点为 800。
          已完成遍历扩展数据块映像。
          LOB 对象总结：总页数为 32 — 已使用的页数为 3
          LBA 对象总结：总页数为 2 — 已使用的页数为 2
         LOB 阶段结束。
       表阶段结束。
    处理已完成。2008-11-20-14.33.57.093001
```

12.5　db2look

12.5.1　db2look 概述

db2look 是可以从命令行提示符下和控制中心中调用的一个强大工具。这个工具可以：

- 从数据库对象中提取数据库定义语言(DDL)语句
- 生成 UPDATE 语句，用于更新数据库管理器和数据库配置参数
- 生成 db2set 命令，用于设置 DB2 概要注册表
- 提取和生成数据库统计报告
- 生成 UPDATE 语句，用于复制关于数据库对象的统计信息

LOAD 之类的实用程序要求目标表已经存在。您可以使用 db2look 提取表的 DDL，在目标数据库上运行它，然后调用装载操作。db2look 非常容易使用，下面的例子展示了这一点。这个命令生成 *peter* 在数据库 department 中创建的所有对象的 DDL，输出被存储在 alltables.sql 中。

```
db2look -d department -u peter -e -o alltables.sql
```

下面的命令生成：

- 数据库 department 中所有对象的 DDL(由-d、-a 和-e 选项指定)
- UPDATE 语句，用于复制数据库中所有表和索引的统计信息(由选项-m 指定)
- GRANT 授权语句(由选项-x 指定)
- 用于数据库管理器和数据库配置参数的 UPDATE 语句和用于概要注册表的 db2set 命令(由选项-f 指定)

```
db2look -d department -a -e -m -x -f -o db2look.sql
```

db2look 还可以生成用于注册 XML 模式的命令。下面的例子生成模式名为 db2inst1 的对象所需的 REGISTER XMLSCHEMA 和 COMPLETE XMLSCHEMA 命令(由选项-xs 指定)。/home/db2inst1 中将创建输出 db2look.sql，这个目录由-xdir 选项指定。

```
db2look -d department -z db2inst1 -xs -xdir /home/db2inst1 -o db2look.sql
```

生成缓冲池、表空间和数据库分区组信息

```
db2look -d <dbname> -l -o storage.out
```

下面是对以上 db2look 命令中所用选项的描述：

- -d：数据库名——该选项必须指定。
- -l：生成数据库布局。这是用于数据库分区组、缓冲池和表空间的布局。
- -o：将输出重新定向到给定的文件名。如果未指定 -o 选项，那么输出将为标准输出(stdout)，通常是输出到屏幕。

创建数据定义语言(DDL)

下列 db2look 命令创建了 DDL 以复制所有数据库对象，以及配置和统计信息：

```
db2look -d <dbname> -e -a -m -o db2look.out
```

这里，我们使用了下列参数：

- -a：为所有的创建器(creator)生成统计数据。如果指定了该选项，那么将忽略-u 选项。
- -e：提取复制数据库所需的 DDL 文件。该选项生成包含了 DDL 语句的脚本。该脚本可以在另一数据库上运行以重新创建数据库对象。
- -m：以模拟模式运行 db2look 实用程序。该选项生成包含了 SQL UPDATE 语句的脚本。这些 SQL UPDATE 语句捕获所有的统计数据。该脚本可以在另一数据库上运行以复制原来的那一个数据库。当指定-m 选项时，将忽略-p、-g 和-s 选项。

收集数据库子集的统计数据和 DDL

为了仅仅收集某些表和相关对象的统计数据和 DDL，可使用下列命令：

```
db2look -d <dbname> -e -a -m -t <table1> <table2> .. <tableX> -o table.ddl
```

这里，我使用了下列附加参数：

- -t：为特定的表生成统计数据。可以将表的最大数目指定为 30。

此外，如果您不使用-a 选项，就可以使用-z 选项：

- -z：模式名。如果同时指定了-z 和-a，那么将忽略-z。

12.5.2　利用 db2look 构建模拟测试数据库

要想构建一个模拟生产环境的测试数据库，我们需要用到 db2look 的下面几个选项：-l 选项导出数据库布局；-m 选项导出数据库统计信息；-f 选项导出数据库配置文件；-fd 选项在测试环境内存资源不足情况下模拟和生产环境同样的内存。

-l 选项

-l 选项对于模拟生产环境十分重要。理想情况下，您需要具有相同的缓冲池、数据库分区组(如果处于多分区环境中)和表空间信息(包括临时表空间)。但是，如果您受到了内存

约束，无法分配生产中所需具有的大型缓冲池，那么就使用 db2fopt 命令。

当然并非总是可以在测试中设置与生产中相同的表空间。例如，可能设置了大型设备，却无法灵活地在测试中创建相同的设备大小。或者，可能根本无法在测试环境中获得单独的表空间设备。此外，或许无法在测试中设置与生产中相同的路径。需要适当地更改路径、设备和文件以适应测试环境。

下面是优化器为表空间所使用的重要信息。这就是您需要确保在测试和生产中相同的信息(**注意：**这里所展示的数字是一个例子，您应在测试中使用与您生产中相同的设置)。

```
PREFETCHSIZE 16
EXTENTSIZE 16
OVERHEAD 12.670000
TRANSFERRATE 0.180000
```

如果生产中表空间是“由数据库管理的”，那么在测试中也应该是“由数据库管理的”。如果它在生产中是“由系统管理的”，那在测试中也应该是这样的方式。

-m 选项

-m 选项极其重要。该选项从系统表收集所有统计数据。测试中的统计数据必须与生产中的相同，这些统计数据是可以在测试环境中模拟生产环境的关键。

-f 和-fd 选项

```
db2look -d <dbname> -f -fd -o config.out
```

- -f：提取配置参数和注册表变量。如果指定了该选项，就会忽略-wrapper 和-server 选项。
- -fd：为 opt_buffpage 和 opt_sortheap 生成 db2fopt 语句，以及其他配置和注册表设置。

该命令的输出如下所示：

```
$ db2look -d sample -f -fd
-- No userid was specified, db2look tries to use Environment variable USER
-- USER is: SKAPOOR
-- This CLP file was created using DB2LOOK Version 8.2
-- Timestamp: Sat Mar 26 00:13:36 EST 2005
-- Database Name: SAMPLE
-- Database Manager Version: DB2/6000 Version 8.2.2
-- Database Codepage: 819
-- Database Collating Sequence is: UNIQUE
CONNECT TO SAMPLE;
```

```
------------------------------------------------------------
-- Database and Database Manager configuration parameters
------------------------------------------------------------
UPDATE DBM CFG USING cpuspeed 6.523521e-07;
UPDATE DBM CFG USING intra_parallel NO;
UPDATE DBM CFG USING federated NO;
UPDATE DBM CFG USING fed_noauth NO;
!db2fopt SAMPLE update opt_buffpage 50000;
!db2fopt SAMPLE update opt_sortheap 10000;
UPDATE DB CFG FOR SAMPLE USING locklist 1000;
UPDATE DB CFG FOR SAMPLE USING dft_degree 1;
UPDATE DB CFG FOR SAMPLE USING maxlocks 10;
UPDATE DB CFG FOR SAMPLE USING avg_appls 1;
UPDATE DB CFG FOR SAMPLE USING stmtheap 2048;
UPDATE DB CFG FOR SAMPLE USING dft_queryopt 5;
-----------------------------------
-- Environment Variables settings
-----------------------------------
!db2set DB2_ANTIJOIN=yes;
!db2set DB2_INLIST_TO_NLJN=yes;
COMMIT WORK;
CONNECT RESET;
TERMINATE;
```

-f 和-fd 选项是用于提取配置参数和注册表变量的关键选项，而优化器将在访问计划阶段使用这些配置参数和环境。在上面的示例中，请注意-fd 选项所产生的下列输出：

```
!db2fopt SAMPLE update opt_buffpage 50000;
!db2fopt SAMPLE update opt_sortheap 10000;
```

db2fopt 命令告诉优化器为“缓冲池大小(buffer pool size)”使用指定的值，而非将可用缓冲池变量的页面加起来。例如，假设由于测试系统上的内存约束，而导致您无法获得大型的缓冲池。您希望将大小配置得相同，实际上却并非有这么大。使用将生成必要的 db2fopt 命令的-fd 选项来告诉优化器使用指定大小，而非基于对该数据库可用的缓冲池进行计算。

12.6　其他工具

12.6.1　db2bfd

db2bfd(DB2 Bind File Description)可以查看静态嵌入 SQL 的绑定文件的头部、宿主变量和时间戳时间，它对我们定位－818 错误非常有帮助。请看下面的例子：

```
 C:\Program Files\IBM\SQLLIB\samples\c>db2bfd -b -s -v tbunion.bnd
----  -b 显示绑定文件头信息   -s 显示 SQL 语句信息  -v 显示变量声明信息
Header Fields:
Field   Value
-----   -----
releaseNum     0x800
Endian  0x4c
numHvars        4
maxSect        26
numStmt        60
optInternalCnt  4
optCount        9
Name                 Value
------------------  -----
Isolation Level     Cursor Stability
Creator             "ORACLE  "
App Name            "TBUNION "     ------程序包名称
Timestamp           "hBzQPULY:2008/11/20 15:16:51:95"  ----绑定时间戳
Cnulreqd            Yes
Sql Error           No package
Validate            Bind
Date               Default/local
Time               Default/local
*** All other options are using default settings as specified by the server ***
tbunion.bnd: SQLStatements = 60  ----------------SQL 语句
Line Sec Typ Var LenSQLstatement text
---- --- --- --- --- ------------------------------------------------------------
  75   0   5   0  21 BEGIN DECLARE SECTION
  80   0   2   0  19 END DECLARE SECTION
 .................很多 SQL 语句，此处略.....................
 695  25   0   0  13 DROP TABLE T2
 697  26   0   0  13 DROP TABLE T3
 700   0   8   0   6 COMMIT
tbunion.bnd: Host Variables = 4
TypeSQLData Type     Length Alias  Name_Len Name           UDT Name
---- ------------    ------ ------ -------- -------------- ---------------
 496 INTEGER              4 H00001       8 prod_num
 460 C STRING          1024 H00004        7 strStmt
```

12.6.2　db2_kill 和 db2nkill

在 UNIX 和 LINUX 上，如果无法正常停止实例，可以使用 db2_kill 来杀掉后台进程，注意做完后最好用 ipcrm 或 ipclean 清除共享内存资源。在 Windows 上对应的命令是

db2nkill。

12.6.3　db2tbst

当表空间出现异常状态时，我们可以使用 **db2tbst** 查看详细的状态信息。看下面的例子：

```
C:\Program Files\IBM\SQLLIB\samples\c>db2tbst -0x0c000000
State = StorDef is in Final State
     + DMS Rebalance in Progress
     + Tablespace Deletion in Progress
     + Tablespace Creation in Progress
```

12.7　本章小结

“君欲善其事、必先利其器”，本章我们给大家讲解了 DB2 中很多功能强大的工具。这些工具能够在某个方面帮我们发现、诊断、分析并辅助解决问题。

第13章

数据库安全

数据库安全在当今社会极其重要。对于一个公司、企业来说，数据库中的数据往往是最重要的资产。在数据库中，这些数据作为商业信息或知识，一旦遭受安全威胁将带来难以想象的严重后果。通常，如果敏感的个人数据(例如手机号码、信用卡号和银行账号等)从不安全的系统中被窃取，那么身份盗用、金融诈骗、未经授权地使用信息等恶果也就接踵而来。因此，系统管理员必须持续监控他们的系统，确保系统中采取了适当的安全预防措施。

在系统架构的不同级别上，可以采用不同的技术来进行安全性控制。例如，可以通过安装防火墙来防止外部网络对服务器的未经授权的访问。可以使用一些安全网络协议技术，例如 IPSec，来保证网络上计算机间通信信道的安全性。又如，可以实行严格的密码策略，要求用户选择一个强密码，并经常更换密码。应用系统的安全需要在操作系统、网络、应用、中间件和数据库等层面相应的安全性防范措施协同工作，这样才能构建一个牢固的安全体系。本章主要讲解和 DB2 数据库有关的安全技术。

本章主要讲解如下内容：

- DB2 安全机制概述
- 认证
- 权限
- 特权
- 某银行安全规划案例
- 执行安全审计
- 基于标签的访问控制
- 安全经验总结

13.1 DB2 安全机制概述

DB2 中有 3 种主要的安全机制，可以帮助 DBA 实现数据库安全计划：身份认证(authentication)、权限(authorization)和特权(privilege)。

身份认证(authentication)

身份认证是用户在尝试访问 DB2 实例或数据库时遇到的第一道安全闸门。身份认证就是使用安全机制验证所提供用户 ID 和口令的过程。用户和密码身份认证由 DB2 外部的设施管理，比如操作系统、域控制器或者 Kerberos 安全系统。这和其他数据库管理系统(DBMS)是不同的，如 Oracle、Informix、Sybase 和 SQL Server，后者既可以在数据库本身定义和验证用户账户，也可在外部设施(如操作系统)中完成。外部安全性服务对希望访问 DB2 服务器的用户进行身份认证，DB2 外部的安全性软件负责处理身份认证。当成功校验了用户 ID 和口令后，内部 DB2 进程将接管控制，并确保用户有权执行所请求的操作。

DB2 数据库是没有用户的，所有用户都是操作系统用户，这和别的数据库不一样。别的数据库有数据库用户和操作系统用户两种类型。而 DB2 中用户使用的是操作系统的用户。之所以有这个限制，是由历史原因造成的。因为 DB2 最初是从 DB2 for MVS/ESA 平台转变过来的，在该平台上有这个限制，所以它的用户创建和认证都要用操作系统来完成。

一旦用户 ID 和口令作为实例附件或数据库连接请求的一部分明确地提供给 DB2，DB2 就会尝试使用该外部安全设施验证用户 ID 和口令。如果请求中没有提供用户 ID 和口令，则 DB2 UDB 隐含使用登录到发出请求的工作站时所用的用户 ID 和口令。

实际的认证位置由 DB2 实例参数 AUTHENTICATION 的值决定。有不同的身份认证方案，包括让用户在 DB2 服务器上认证(使用服务器的安全设施)、在客户机上认证(允许“单点登录”访问)、使用 Kerberos 安全设施认证，或者用户定义的通用安全服务(Generic Security Service，GSS)插件认证。其他身份认证选项还包括：当用户名和口令以及数据在客户机和服务器之间的网络上传递时进行加密。为 AUTHENTICATION 参数选择的值依赖于具体环境和本地安全策略。

权限(authorization)

权限涉及将 DB2 角色赋予用户和/或组。每一种角色具有一定级别的权限，对特定数据库和/或其中的对象执行某些命令。DB2 中包括以下 7 种不同角色或权限：SYSADM、SYSCTRL、SYSMAINT、SYSMON、DBADM、SECADM 和 LOAD。

特权(privilege)

特权的粒度比授权要细，可以分配给用户和/或用户组。特权定义用户可以创建或删除的对象。它们还定义用户可以用来访问对象(比如表、视图、索引和应用程序包)的命令。DB2 V9 还新增了一个概念——基于标签的访问控制(LBAC)，它允许以更细的粒度控制谁有权访问单独的行和/或列。

安全层次

图 13-1 说明了 DB2 的安全层次结构，当一个用户在客户端发出一条“select * from account”SQL 语句时，首先要连接数据库，在连接数据库时需要提供用户名和密码。用户名和密码是在数据库之外认证的，这需要用到操作系统或外部安全插件。例如在 AIX 上是使用/etc/passwd 验证用户名，使用/etc/security/passwd 来验证密码。这是操作系统层次。

操作系统之下是实例级别的权限，实例之下是数据库级别的权限，用户要访问表 account，也必须在表 account 上有特定的 select 特权。这就是整个 DB2 的安全机制实现方式。

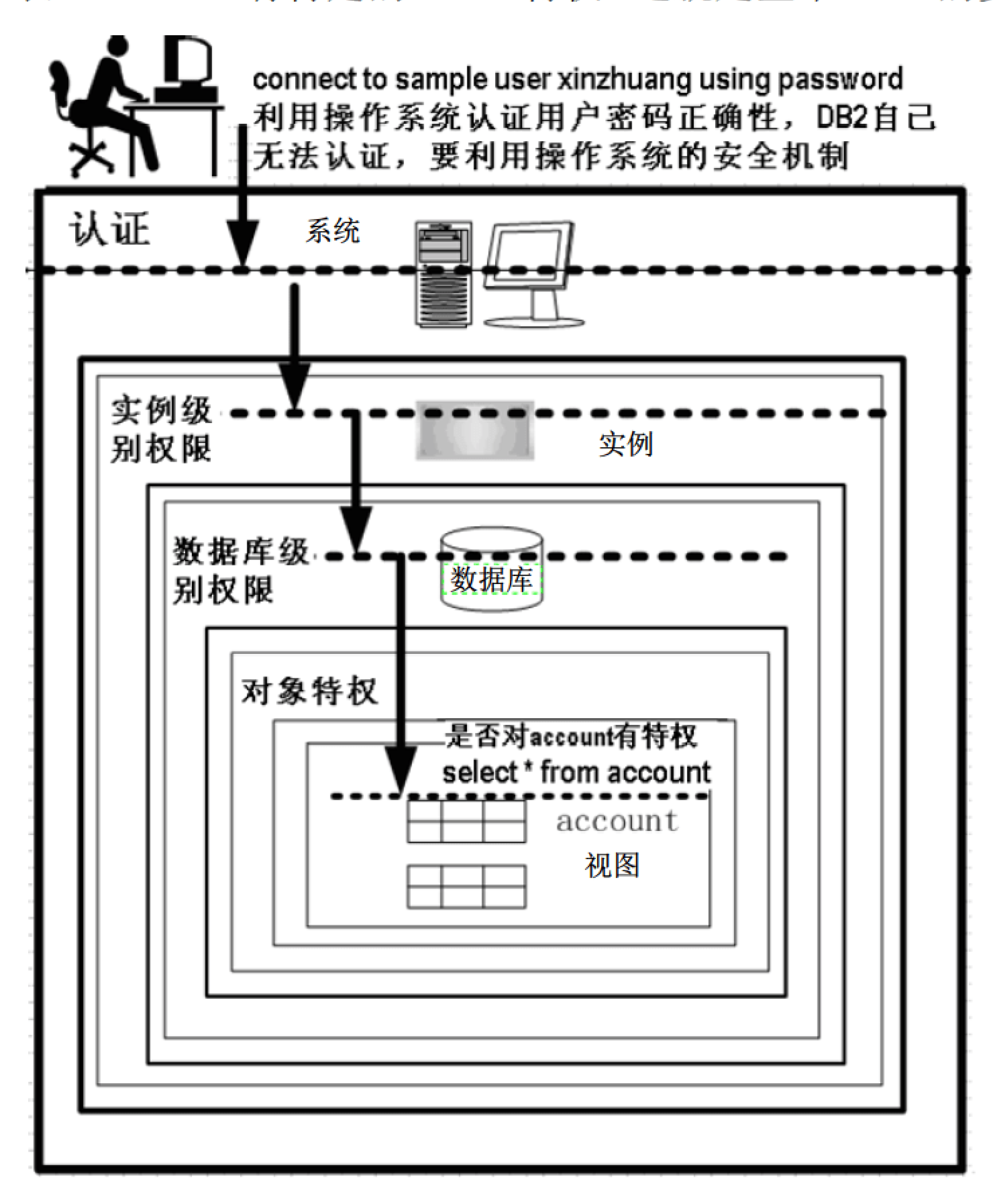


图 13-1　DB2 安全层次结构

下面我们用一个例子来说明 DB2 的安全模型实现机制。比如，图 13-2 中的连接语句供用户 *bob* 使用口令 *bobpsw* 连接到 *finance* 数据库。身份认证过程包括下面 7 个步骤：

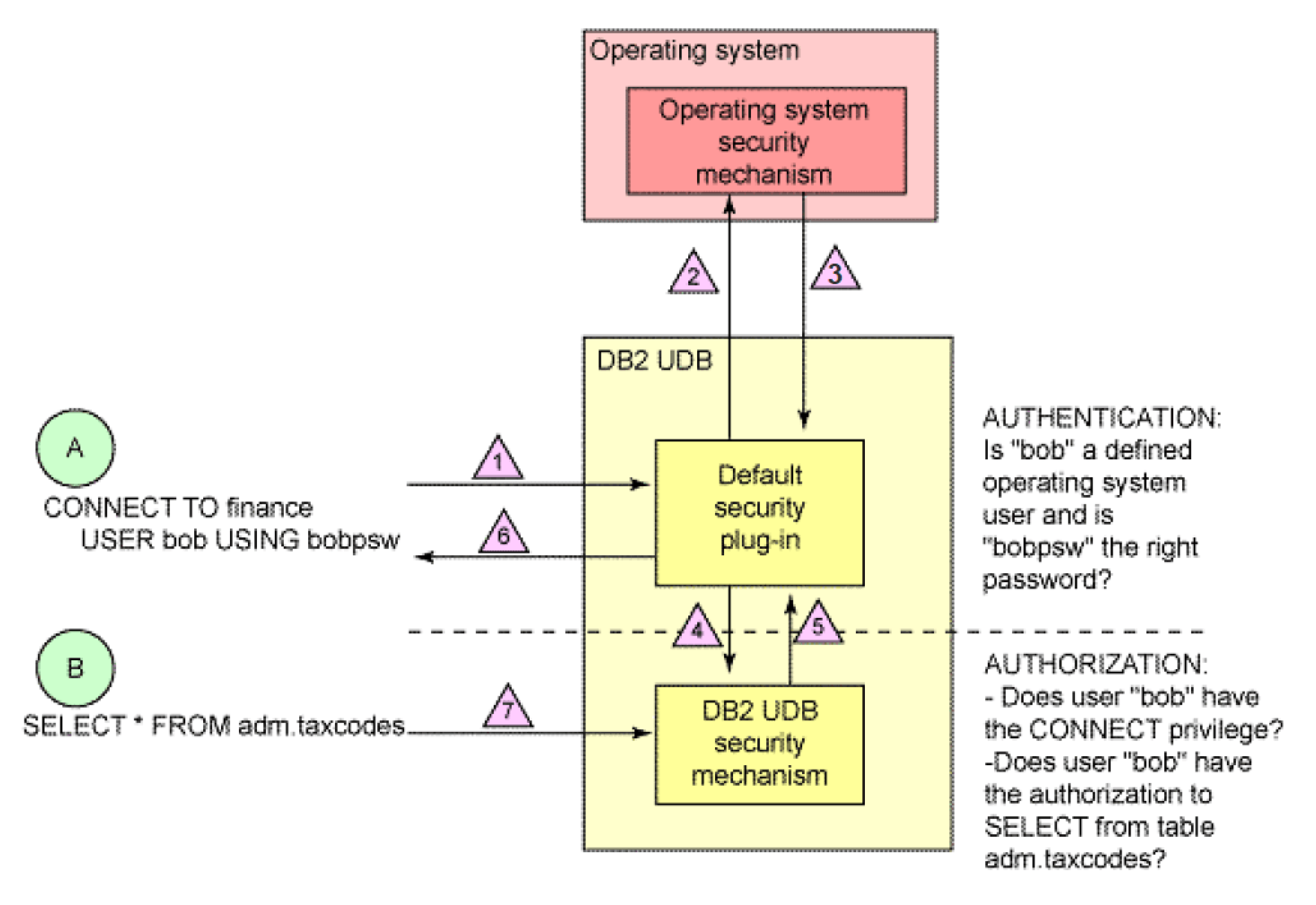


图 13-2　DB2 安全模型实现机制

(1) CONNECT 语句传递给 DB2 数据库服务器。

(2) 如果没有明确配置安全插件，则使用默认的安全插件。如果包含 *finance* 数据库的实例的 AUTHENTICATION 参数设为 SERVER(默认设置)，连接请求中的用户 ID 和口令则由 DB2 数据库服务器上的安全设施验证。默认插件将用户 ID 和口令发送给操作系统进行验证。

(3) 操作系统确认 *bob/bobpsw* 组合是否有效，把该信息返回给安全插件。

(4)安全插件激活 DB2 安全机制，对用户 *bob* 查询 DB2 系统编目表中和权限相关的表，看看该用户是否被授予了该数据库的 CONNECT 权限。默认情况下，CONNECT 特权被授予 PUBLIC，就是说任何通过身份认证的用户都能连接数据库。

(5) DB2 安全机制验证用户 *bob*，并且把成功或者错误信息返回给安全插件。

(6)安全插件把成功或者失败的消息返回给用户。如果用户没有通过身份认证，DB2 就会拒绝连接请求，并向客户机应用程序返回错误消息，如下所示：

```
SQL30082N  Attempt to establish connection failed with security
reason "24"("USERNAME AND/OR PASSWORD INVALID"). SQLSTATE=08001
```

这时，DB2 服务器上的 DB2 诊断日志(db2diag.log)中也会出现类似于下面这样的记录：

```
2008-07-09-16.18.33.546000-240 I729347H256       LEVEL: Severe
PID     : 3888                TID : 604
FUNCTION:DB2Common, Security, Users and Groups, secLogMessage, probe:20
DATA #1 : String, 44 bytes
check password failed with rc = -2146500502
```

如果遇到这样的消息，一定要确认连接到数据库的用户或应用程序是否提供了合法的用户 ID 和口令。该用户 ID 和口令必须存在于执行用户身份认证的设施中(由目标 DB2 UDB 实例的 AUTHENTICATION 参数决定)。

13.2　认证(authentication)

13.2.1　什么时候进行 DB2 身份认证

DB2 身份认证控制数据库安全性策略的以下方面：

- 谁有权访问实例和/或数据库
- 在哪里以及如何检验用户的密码

在发出 *attach* 或 *connect* 命令时，**它借助于底层操作系统的安全特性实现对 DB2 用户的身份认证。**attach 命令用来连接 DB2 实例，而 connect 命令则用来连接 DB2 实例中的数据库。下面的示例展示了 DB2 对发出这些命令的用户进行身份认证的不同方式，这些示例在数据库管理程序配置文件中使用默认的身份认证类型 SERVER。最后的一个示例说明了如何使用 DB2 修改服务器操作系统上的密码。

用创建 DB2 实例时使用的用户 ID 登录到安装 DB2 的机器上。发出以下命令：

```
db2 attach to DB2
```

在这里，虽然没有显式地提供用户名和密码，但是隐式地执行了身份认证。使用登录机器的用户 ID，并假设这个 ID 和密码已经经过了操作系统的检验。

```
db2 connect to sample user test1 using password ibmdb2
Database server        = DB2/NT 9.5.0
SQL authorizationID  = TEST1
Local database alias   = SAMPLE
```

在这里，显式地执行了身份认证。用户 *test1* 和密码 *password* 由操作系统进行检验。用户 *test1* 成功地连接到示例数据库。

```
db2 connect to sample user test1 using password new chgpass confirm chgpass
```

与前面的第 2 个示例一样，用户 ID *test1* 和密码 *password* 由操作系统进行检验。然后，操作系统将 *test1* 的密码从 *password* 改为 *chgpass*。因此，如果再次发出前面的第 2 个示例的命令，它就会失败。

13.2.2　DB2 身份认证类型

1. 客户机、服务器

在考虑整个 DB2 数据库环境的安全性时，理解术语客户机、服务器是相当重要的。数据库环境常常由几台不同的机器组成，必须在所有潜在的数据访问点上保护数据库。在处理 DB2 身份认证时，理解客户机、服务器的概念尤其重要。

图 13-3 说明了基本的客户机-服务器配置。



图 13-3　客户机与服务器

数据库服务器是数据库实际所在的机器(在分区的数据库系统上可能是多台机器)。DB2 数据库客户机是对服务器上的数据库执行查询的机器。这些客户机可以是本地的(驻留在与数据库服务器相同的物理机器上)，也可以是远程的(驻留在单独的机器上)。

DB2 在实例级使用了一个身份认证类型参数 AUTHENTICATION，这个参数决定了在什么地方进行身份认证。例如，在客户机-服务器环境中，AUTHENTICATION 参数的设置决定了是在客户机上还是在服务器上检验用户的 ID 和密码。

2. 身份认证类型

DB2 V9 能够根据用户是试图连接数据库，还是执行实例连接和实例级操作，指定不同的身份认证机制。在默认情况下，实例对于所有实例级和连接级请求使用一种身份认证类型。这由数据库管理程序配置参数 AUTHENTICATION 来指定。DB2 V9.1 中引入了数据库管理程序配置参数 SRVCON_AUTH。这个参数专门处理对数据库的连接。例如，如果在 DBM CFG 中进行以下设置：

```
DB2 GET DBM CFG
Server Connection Authentication         (SRVCON_AUTH)  = KERBEROS
Database manager authentication        (AUTHENTICATION)  = SERVER_ENCRYPT
```

那么在连接实例时会使用 SERVER_ENCRYPT。但是在连接数据库时会使用 KERBEROS 身份认证。如果在服务器上没有正确地初始化 KERBEROS，但是提供了有效的用户 ID/口令，那么允许这个用户连接实例，但是不允许他连接数据库。身份认证类型确定在何处验证用户 ID/口令对。所支持的主要身份认证类型有：

- SERVER(默认)
- SERVER_ENCRYPT
- KERBEROS
- KRB_SERVER_ENCRYPT
- CLIENT

身份认证类型是在服务器和客户机处同时设置的。

服务器

每个实例仅允许一种类型的身份认证，也就是说，设置适用于该实例下定义的所有数据库。在数据库管理器配置文件中使用 AUTHENTICATION 参数指定该设置。

```
db2 update database manager configuration authentication auth_type
```

客户机

在客户机上编目的各数据库拥有自己的身份认证类型，使用 catalog database 命令指定。

```
db2 catalog database db_name at node node_name authentication auth_type
```

1) 使用 SERVER 选项进行身份认证

使用 SERVER 选项时，用户 ID 和口令将发送到服务器进行校验。考虑以下示例。

(1) 用户使用用户名 *peter* 和口令 *peterpwd* 登录到工作站。

(2) *peter* 随后使用用户 ID*db2user* 和口令 *db2pwd* 连接到 SAMPLE 数据库，这是在远程 DB2 服务器上定义的。

(3) *db2user* 和 *db2pwd* 通过网络发送到服务器。

(4) *db2user* 和 *db2pwd* 在 DB2 服务器上校验。

整个过程如图 13-4 所示。

若您想避免用户 ID 和口令在网络上被窃听，可使用 SERVER_ENCRYPT 身份认证类型，这样用户 ID 和口令就都会被加密。

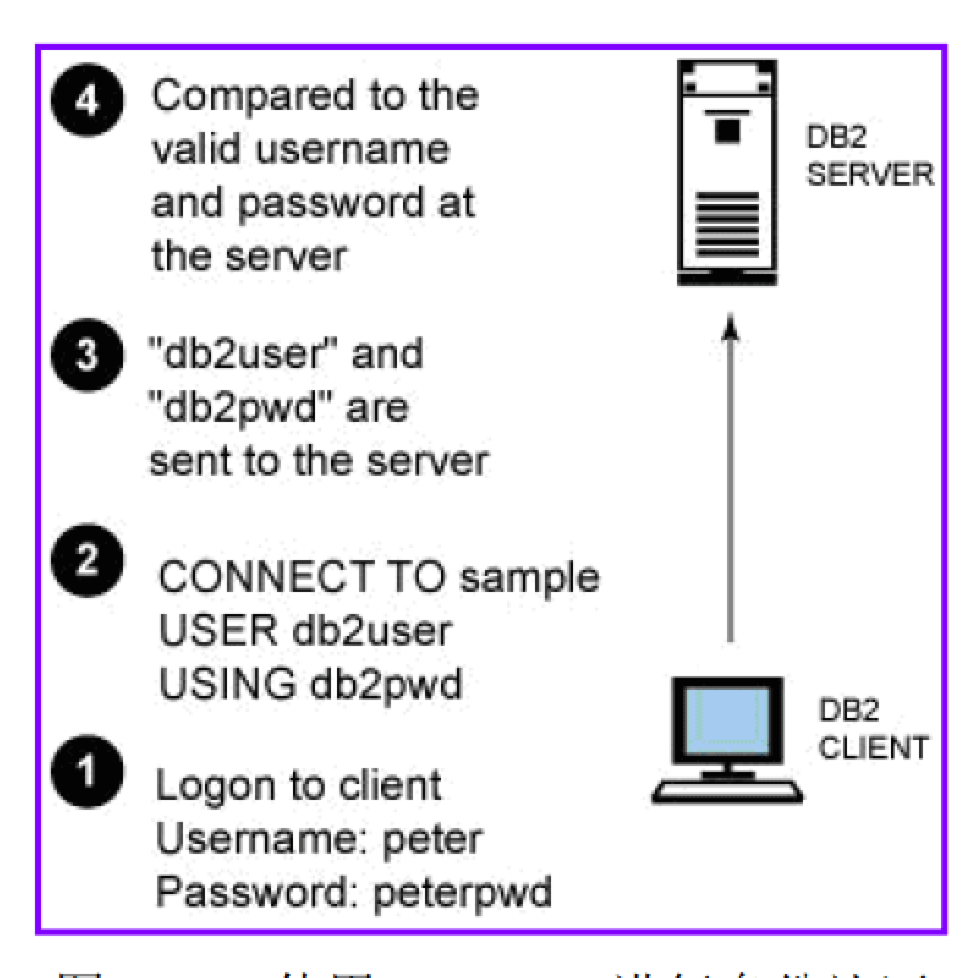


图 13-4　使用 SERVER 进行身份认证

2) 使用 Kerberos 进行身份认证

Kerberos 是一种外部安全性协议，它使用通用密码术创建共享的加密密钥。Kerberos 安全协议作为第三方身份认证服务执行身份认证，它使用传统的密码术创建一个共享的密钥。这个密钥成为用户的凭证，在请求本地或网络服务时在所有情况下都使用它检验用户的身份。Kerberos 提供了安全的身份认证机制，这是因为用户 ID 和口令不再需要以明文形式通过网络传输。通过使用 Kerberos 安全协议，可以实现对远程 DB2 数据库服务器的单点登录。

图 13-5 展示了 Kerberos 身份认证在 DB2 中的工作原理。

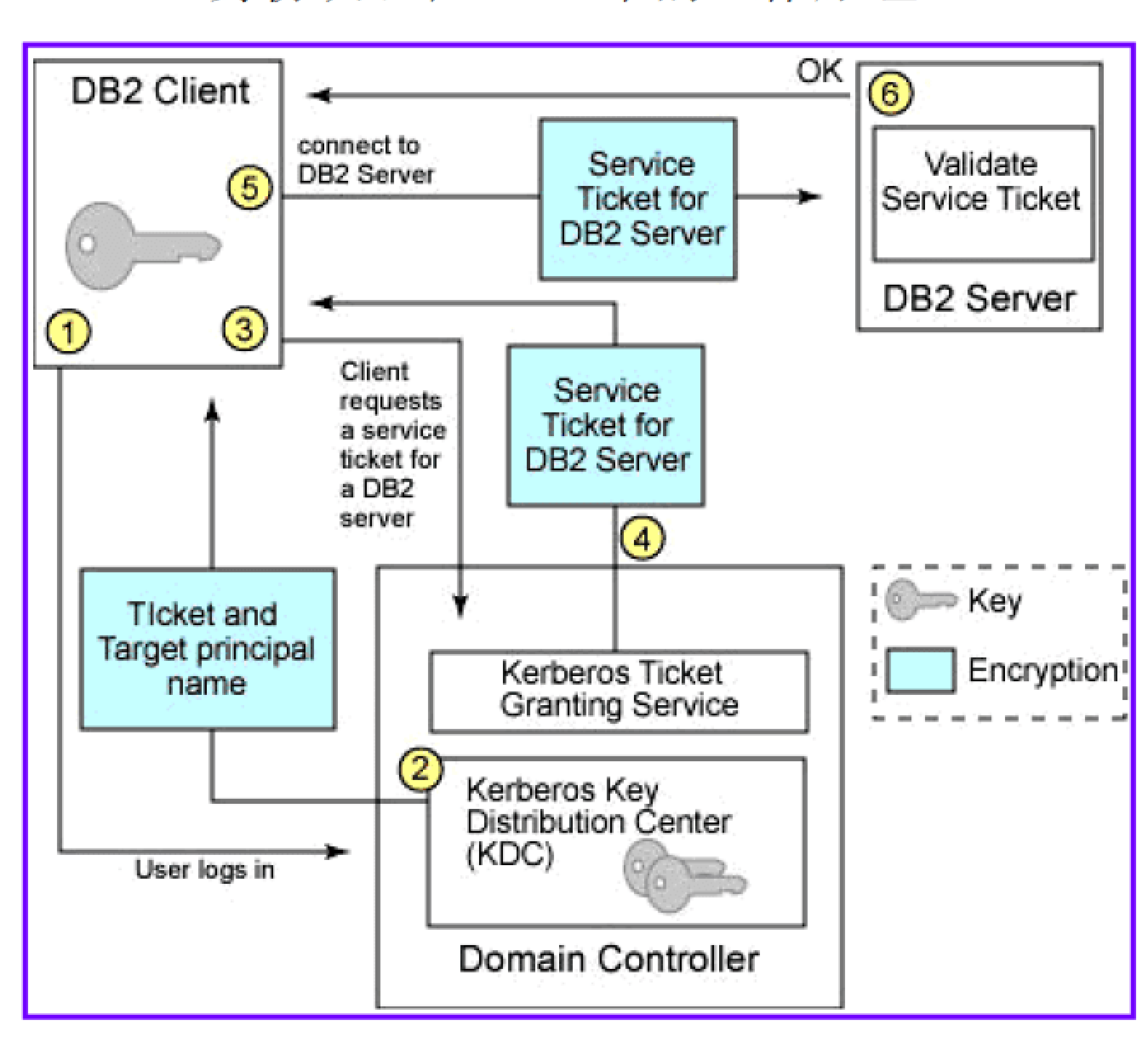


图 13-5　使用 kerberos 进行身份认证

如上所述，Kerberos 身份认证在 DB2 中是使用插件架构实现的。默认的 Kerberos 插件的源代码在 samples/security/plugins 目录中，称为 IBMkrb5.c。在 DB2 中使用 Kerberos 之前，必须在客户机和服务器上启用和支持 Kerberos。为此，必须满足以下条件：

- 客户机和服务器必须属于同一个域(用 Windows 术语来说，是可信域)
- 必须设置适当的主体(Kerberos 中的用户 ID)
- 必须创建服务器的 keytab 文件，实例所有者必须能够读这个文件
- 所有机器必须有同步的时钟

DB2 客户机和服务器均支持 Kerberos 安全协议时，即可使用 Kerberos 身份认证类型。某些客户机可能并不支持 Kerberos，但依然需要访问 DB2 服务器。为确保所有类型的客户机均能安全地连接，将 DB2 服务器的身份认证类型设置为 KRB_SERVER_ENCRYPT。这将允许所有启用了 Kerberos 的客户机使用 Kerberos 进行身份认证，而其他客户机则使用

SERVER_ENCRYPT 身份认证，如图 13-6 所示。

Client Specification	Server Specification	Client/Server Resolution
KERBEROS	KRB_SERVER_ENCRYPT	KERBEROS
Any other setting	KRB_SERVER_ENCRYPT	SERVER_ENCRYPT

图 13-6　确保所有客户机均能安全连接

设置 Kerberos 身份认证

为了在 DB2 中启用 Kerberos 身份认证，必须先告诉客户机在哪里寻找将使用的 Kerberos 插件。在客户机上，运行以下命令：

```
DB2 UPDATE DBM CFG USING CLNT_KRB_PLUGIN IBMkrb5
```

在上面的示例中，使用默认的 Kerberos 插件。如果使用的 Kerberos 需要实现特殊功能，DBA 可以通过修改这个插件来执行特殊功能。

还可以告诉客户机它正在针对哪个服务器主体进行身份认证。这个选项可以避免 Kerberos 身份认证的第一步，即客户机寻找它要连接的实例所在的服务器主体。在客户机上对数据库进行编目时可以指定 AUTHENTICATION 参数。它的格式是：

```
DB2 CATALOG DB dbname AT NODE N1 AUTHENTICATION KERBEROS TARGET PRINCIPAL
   service/host@REALM
```

下面这一步是可选的：

```
DB2 CATALOG DB sample AT NODE N1 AUTHENTICATION KERBEROS TARGET PRINCIPAL
   gmilne/NIUXINZHUANG.ZH@CCB.COM
```

设置 Kerberos 身份认证之后的下一步是设置服务器。srvcon_gssplugin_list 参数可以设置为支持的 GSS-API 插件的列表，但是该列表中最多只允许包含一个 Kerberos 插件。如果这个列表中没有 Kerberos 插件，并且 AUTHENTICATION 参数被设置为 Kerberos 或 KRB_SVR_ENCRYPT，那么将自动使用默认的 IBMkrb5 插件。如果希望允许所有身份认证(实例连接和数据库连接)使用 Kerberos，那么执行以下命令：

```
DB2 UPDATE DBM CFG USING AUTHENTICATION KERBEROS
```

或者

```
DB2 UPDATE DBM CFG USING AUTHENTICATION KRB_SERVER_ENCRYPT
```

如果只希望 DB2 使用 Kerberos 对数据库连接进行身份认证(对实例连接使用 SERVER)，那么执行以下命令：

```
DB2 UPDATE DBM CFG USING SVRCON_AUTH KERBEROS                    or
```

或者

```
DB2 UPDATE DBM CFG USING SVRCON_AUTH KRB_SERVER_ENCRYPT
```

根据实例的位长度(32 或 64 位)，DB2 将在实例启动时自动地装载 IBMkrb5 插件。

3) 在客户机上进行身份认证

这一选项允许在客户机上进行身份认证。用户成功登录到客户机后，即可轻松连接到数据库，而无需再次提供口令，如图 13-7 所示。

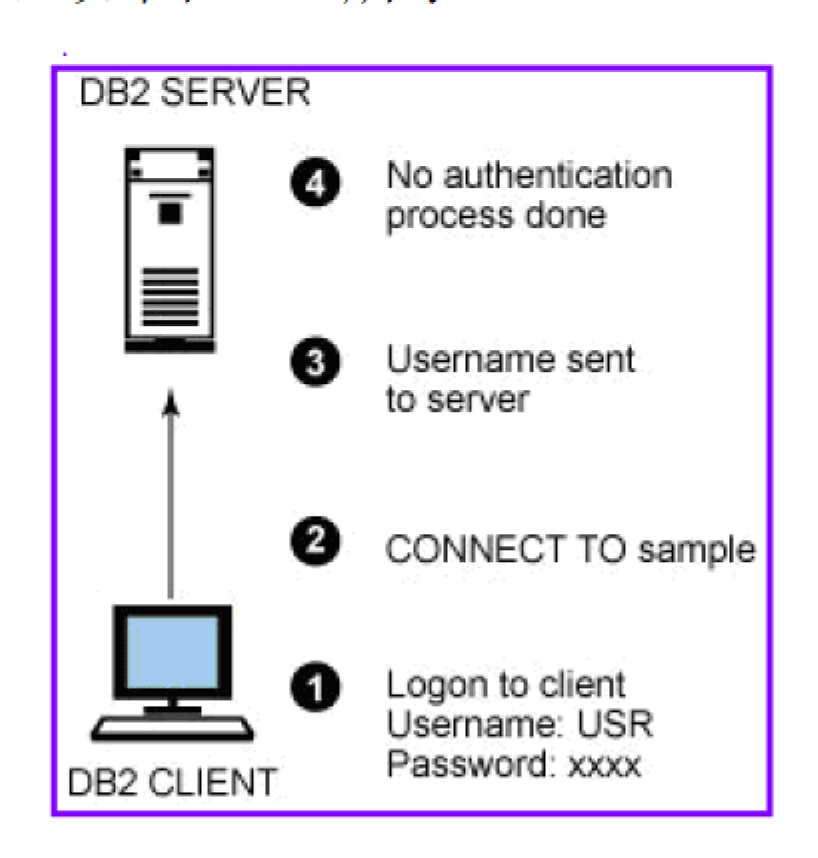


图 13-7　在客户机上进行身份认证

这里有一个重要问题需要考虑：有些客户机系统不具有可靠的安全性设施，例如 Windows 9x 和 Classic Mac OS。它们被称作不受信任的客户机。任何人只要可以访问这些系统，就可以不经过身份认证直接连接到 DB2 服务器。谁知道它们会执行怎样的破坏性操作(例如删除一个数据库)？为实现允许受信任的客户机自行执行身份认证，同时强制不受信任的客户机在服务器处进行身份认证的灵活性，DB2 引入了另外两个数据库管理器配置参数：

- TRUST_ALLCLNTS
- TRUST_CLNTAUTH

这两个参数仅在身份认证设置为 CLIENT 时生效。

信任客户机

TRUST_ALLCLNTS 确定信任哪种类型的客户机。该参数有以下 3 种可能值：

- YES——信任所有客户机。这是默认设置。身份认证将在客户机处执行。但有一个例外，我们将在介绍 TRUST_CLNTAUTH 时对此予以详细的讨论。
- NO——仅信任具备可靠的安全性设施的客户机(受信任的客户机)。对于不受信任的客户机连接，则必须提供用户 ID 和口令，以便在服务器进行身份认证。
- DRDAONLY——仅信任在 iSeries 或 zSeries 平台上运行的客户机(例如 DRDA 客户机)。其他任何客户机都必须提供用户 ID 和口令。

设想一个场景，DB2 服务器将身份认证设置为 CLIENT，TRUST_ALLCLNTS 设置为 YES。您作为 *localuser* 登录到一台 Windows XP 计算机，并在未指定用户 ID 和口令的情况下连接到远程数据库。那么 *localuser* 将成为数据库处的连接授权 ID。而如果您想使用其他用户 ID 连接到数据库(例如，有执行数据库备份权限的 *poweruser*)，又会怎么样呢?

为允许此类行为，可使用 TRUST_CLNTAUTH 来指定，当在 connect 语句或 attach 命令中提供了用户 ID 和密码时将在何处进行身份认证。允许使用的值有两个：

- CLIENT——身份认证在客户机处执行，不需要用户 ID 和口令。
- SERVER——身份认证在服务器处完成，需要提供用户 ID 和口令。

下面的示例说明如何在服务器和客户机上设置身份认证类型和参数：

在服务器上设置身份认证：

```
db2 update dbm cfg using authentication client
db2 update dbm cfg using trust_allclnts yes
db2 update dbm cfg using trust_clntauth server ---重启实例 db2start 以生效
```

在客户机上设置身份认证：

```
db2 catalog database sample at node nd1 authentication client
```

在上面的示例中，如果从任何客户机发出如下命令：

```
db2 connect to sample
```

那么身份认证在客户机上进行。

如果从任何客户机发出如下命令：

```
db2 connect to sample user test1 using password
```

那么身份认证在服务器上进行。

4) 其他身份认证设置

如果查看 DB2 V9.1 实例中的 DBM CFG，会看到影响 DB2 对用户 ID 进行身份认证的方式的各种设置。正如前面提到的，有针对标准操作系统用户 ID 身份认证的设置

(CLIENT、SERVER、SERVER_ENCRYPT、DATA_ENCRYPT、DATA_ENCRYPT_CMP)，还有将身份认证工作交给外部程序的插件(KERBEROS、KRB_SERVER_ENCRYPT、GSSPLUGIN、GSS_SERVER_ENCRYPT)。下面专门介绍其他一些配置变量如何影响对用户的身份认证。

```
Client Userid-Password Plugin          (CLNT_PW_PLUGIN) =
Group Plugin                          (GROUP_PLUGIN) =
GSS Plugin for Local Authorization    (LOCAL_GSSPLUGIN) =
Server Plugin Mode                   (SRV_PLUGIN_MODE) = UNFENCED
Server List of GSS Plugins     (SRVCON_GSSPLUGIN_LIST) =
Server Userid-Password Plugin        (SRVCON_PW_PLUGIN) =
Cataloging allowed without authority  (CATALOG_NOAUTH) = NO
Bypass federated authentication           (FED_NOAUTH) = NO
```

在下面的列表中，不包括已经讨论过的参数。

- CLNT_PW_PLUGIN：这个参数在客户端的 DBM CFG 中指定。它指定用来进行客户机和本地身份认证的客户机插件的名称。
- GROUP_PLUGIN：默认值是空(NULL)。将它设置为用户定义的插件，就会调用这个插件进行所有组枚举，而不依赖于操作系统的组查找。这与后面讨论的授权部分相关。
- LOCAL_GSSPLUGIN：这个参数指定默认的 GSS-API 插件库的名称，当数据库管理程序配置的身份认证参数值设置为 GSSPLUGIN 或 GSS_SERVER_ENCRYPT 时，这个库用来进行实例级的本地授权。
- SRV_PLUGIN_MODE(YES/NO)：这个参数的默认设置是 NO。当改为 YES 时，以 FENCED 模式启动 GSS-API 插件，其工作方式类似于 FENCED 存储过程。FENCED 插件的崩溃不会导致 DB2 实例崩溃。在插件还处于开发阶段时，建议以 FENCED 模式运行它们，这样的话插件中的逻辑问题和内存泄漏就不会导致实例崩溃。在确定插件是可靠的之后，应该以 UNFENCED 模式运行以提高性能。
- SRVCON_GSSPLUGIN_LIST：一个插件列表，当使用 KERBEROS、KRB_SERVER_ENCRYPT、GSSPLUGIN 或 GSS_SERVER_ENCRYPT 时，服务器上的数据库管理程序在身份认证期间将使用这些插件。列表中的每个插件应该用逗号(，)分隔，它们之间没有空格。插件按照优先次序列出，首先使用列表中的第一个插件对发送的用户 ID/口令进行身份认证。只有当列出的所有插件都返回了错误时，DB2 才会向用户返回身份认证错误。
- SRVCON_PW_PLUGIN：在指定 CLIENT、SERVER 或 SERVER_ENCRYPT 身份认证时，这个参数允许用户修改 DB2 用来检验用户 ID 和密码的默认身份认证方法。

在默认情况下，它的值是 NULL，因此使用默认的 DB2 方法。

- CATALOG_NOAUTH(YES/NO)：默认值是 NO。将这个参数修改为 YES 就允许不属于 SYSADM、SYSCTRL 或 SYSMAINT 组成员的用户修改机器上的 Database、Node、Admin 和 DCS 编目。登录的用户使用不可信客户机，或者登录所用的用户 ID 不允许连接数据库或实例，但是用户又必须在客户机上进行编目，只有在这种情况下这个参数才是有用的。
- FED_NOAUTH：如果 FED_NOAUTH 设置为 YES，身份认证设置为 SERVER 或 SERVER_ENCRYPT，联邦设置为 YES，那么在实例上避免进行身份认证。它假设身份认证在数据源上进行。当 FED_NOAUTH 设置为 YES 时系统会发出警告。在客户机和 DB2 服务器上都不进行身份认证。知道 SYSADM 身份认证名称的任何用户都拥有联邦服务器的 SYSADM 权限。

13.3　权限(authorization)

13.3.1　权限层次

DB2 定义了一个权限层次结构，用于将一组预先确定的管理权限授予用户账号组。这些管理权限包括能够对数据库进行备份、更改配置参数、查看表数据等等。权限级别控制执行数据库管理器维护操作和管理数据库对象的能力。

DB2 授权控制数据库安全策略的以下方面：

- 用户被授予的权限级别
- 允许用户运行的命令
- 允许用户读取和/或修改的数据
- 允许用户创建、修改和/或删除的数据库对象

按照权限作用范围来区分，DB2 中共包括两类权限：实例级和数据库级权限。在实例级上定义的权限会应用于这个实例中的所有数据库上；而在数据库级上定义的权限则仅应用到特定的数据库，对同一个实例中的其他数据库不会产生影响。

具体来说，实例级权限包括 SYSADM、SYSCTRL、SYSMAINT、SYSMON 等。这意味着上述 4 种权限的作用范围包括实例级命令以及针对这个实例中的所有数据库的命令。这些权限只能分配给组，可以通过 DBM CFG 文件分配这些权限。而数据库级上定义的权限则包括 DBADM、SECADM 和 LOAD 这 3 种权限，针对特定数据库的 DBADM、SECADM 和 LOAD 权限可以分配给用户或用户组。可以使用 GRANT 命令显式地分配这些权限。

注意，本章中任何提到组成员关系的地方都假设在操作系统级上已经定义了这些用户

和组名。

用户可以通过发出以下命令来判断自己拥有哪些权限和数据库级特权：

```
db2 get authorizations
```

权限级别按照图 13-8 所示的层次结构进行组织。这个层次结构的顶部是 SYSADM 权限级别，这是用户在 DB2 中可以拥有最高权限级别。具有 SYSADM 权限的用户可以执行所有可用的 DB2 操作。SYSCTRL 和 SYSMAINT 权限级别提供了 SYSADM 权限的子集，可以管理系统，但是不允许访问表中的任何数据。SYSMON 权限提供了使用数据库系统监视器的能力。DBADM 权限允许用户在一个实例的特定数据库上执行管理任务，还允许访问这个数据库中所有的数据和对象。LOAD 权限允许用户运行 LOAD 实用程序，这是 DB2 UDB 的高速批量数据装载器。SECADM 权限用于管理一个或多个数据库中的安全性。

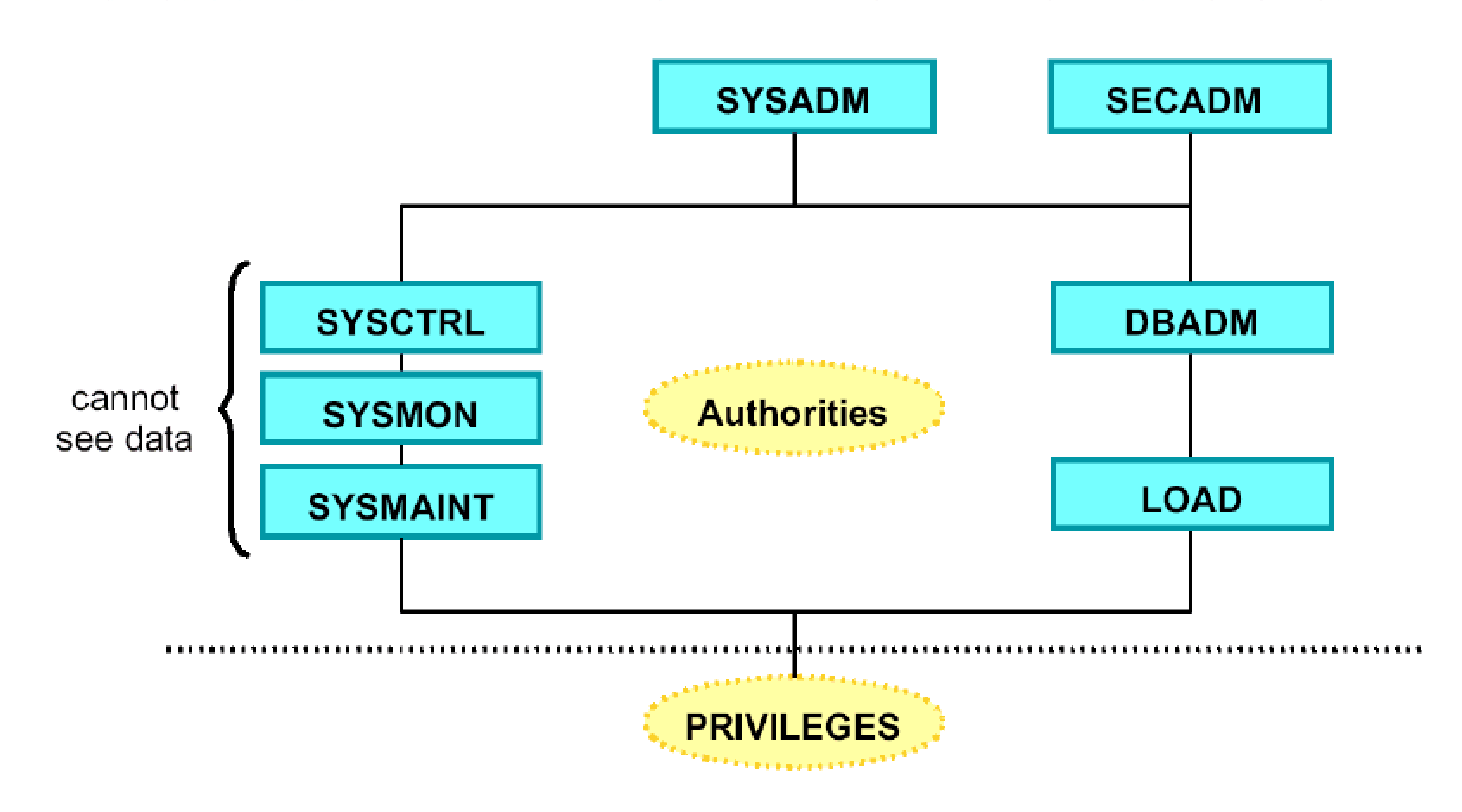


图 13-8　DB2 的权限级别

表 13-1 总结了每个权限的级别及用途。

表 13-1　权限级别及用途总结

权限级别	说明和用途
SYSADM	DB2 UDB 中最高的管理权限级别。具有 SYSADM 权限的用户可以运行实用程序，发出数据库和数据库管理器命令，以及访问这个数据库管理器实例中任何数据库中任何表中的数据 提供控制这个实例中所有数据库对象的能力，包括数据库、表、视图、索引、包、模式、服务器、别名、数据类型、函数、过程、触发器、表空间、数据库分区组、缓冲池和事件监视器 该权限供需要对实用程序和数据的完全访问权的 DB2 UDB 管理员使用

(续表)

权限级别	说明和用途
SYSCTRL	最高的系统控制权限级别。提供对数据库管理器实例及其数据库执行维护和管理操作的能力 不允许直接访问数据库中的数据。具有连接数据库的隐式特权，并可以执行具有 SYSMAINT 和 SYSMON 权限的用户能够执行的功能。该权限供管理一个包含敏感数据的数据库管理器实例的用户使用
SYSMAINT	次高的系统控制权限级别。提供对数据库管理器实例及其数据库执行维护和管理操作的能力 不允许直接访问数据库中的数据。具有连接数据库的隐式特权，并可以执行具有 SYSMON 权限的用户能够执行的功能。该权限供维护一个包含敏感数据的数据库管理器实例中的数据库的用户使用
SYSMON	提供获得数据库管理器实例及其数据库的快照的能力。如果一个数据库管理器实例中的数据库包含敏感数据，而管理用户只需要通过快照监控数据来进行问题判断，这时候就可以给该用户授予 SYSMON 权限。该权限不允许改变系统资源的使用
DBADM	对于一个实例中的一个特定数据库的次高管理权限级别。允许用户运行某些实用程序，发出数据库命令以及访问数据库中任何表中的数据。该权限供需要完全访问数据库对象和数据，但是不需要完整的维护权限的管理员使用
LOAD	允许用户调用 LOAD 实用程序。根据 LOAD 操作的模式，用户还需要 LOAD 操作目标表上的 INSERT 和 DELETE 特权。该权限供只想批量装载一组新数据的用户使用

表 13-2 对比了每个权限级别允许的常见管理操作。

表 13-2　每个权限级别允许的常见管理操作的比较

功　　能	SYSADM	SYSCTRL	SYSMAINT	SYSMON	DBADM	LOAD
MIGRATE DATABASE	YES	NO	NO	NO	NO	NO
GRANT/REVOKE DBADM	YES	NO	NO	NO	NO	NO
UPDATE DBM CFG	YES	NO	NO	NO	NO	NO
ESTABLISH/CHANGE SYSCTRL/SYSMAINT AUTHORITY	YES	NO	NO	NO	NO	NO

(续表)

功　　能	SYSADM	SYSCTRL	SYSMAINT	SYSMON	DBADM	LOAD
UPDATE DB/NODE/DCS DIRECTORIES	YES	YES	NO	NO	NO	NO
FORCE USERS OFF DATABASE	YES	YES	NO	NO	NO	NO
CREATE/DROP DATABASE	YES	YES	NO	NO	NO	NO
CREATE/DROP/ALTER TABLE SPACE	YES	YES	NO	NO	NO	NO
RESTORE TO NEW DATABASE	YES	YES	NO	NO	NO	NO
UPDATE DB CFG	YES	YES	YES	NO	NO	NO
BACKUP DATABASE OR TABLE SPACE	YES	YES	YES	NO	NO	NO
RESTORE TO EXISTING DATABASE	YES	YES	YES	NO	NO	NO
PERFORM ROLLFORWARD RECOVERY	YES	YES	YES	NO	NO	NO
START/STOP DATABASE INSTANCE	YES	YES	YES	NO	NO	NO
RESTORE TABLE SPACE	YES	YES	YES	NO	NO	NO
RUN TRACE	YES	YES	YES	NO	NO	NO
OBTAIN MONITOR SNAPSHOTS	YES	YES	YES	YES	NO	NO
CREATE/ACTIVATE/DROP EVENT MONITOR	YES	NO	NO	NO	YES	NO
QUERY TABLE SPACE STATE	YES	YES	YES	NO	YES	YES

(续表)

功　　能	SYSADM	SYSCTRL	SYSMAINT	SYSMON	DBADM	LOAD
PRUNE LOG HISTORY FILES	YES	YES	YES	NO	YES	NO
QUIESCE INSTANCES	YES	YES	NO	NO	NO	NO
QUIESCE DATABASES	YES	NO	NO	NO	YES	NO
QUIESCE TABLE SPACE	YES	YES	YES	NO	YES	YES
REORG TABLE	YES	YES	YES	NO	YES	NO
RUN RUNSTATS UTILITY	YES	YES	YES	NO	YES	YES
LOAD TABLE	YES	NO	NO	NO	YES	YES
READ DATABASE TABLE DATA	YES	NO	NO	NO	YES	NO

13.3.2　授予/撤销实例级权限

获得实例级权限的办法是，将在外部安全设施中定义的用户组赋给相关的实例级权限参数(SYSADM_GROUP、SYSCTRL_GROUP、SYSMAINT_GROUP、SYSMON_GROUP)。例如，如果希望用户账号 xinzhuang 具有 SYSMAINT 权限，那么可以将 xinzhuang 放进 MAINT 组，然后将实例参数 SYSMAINT_GROUP 更新为 MAINT。这样，MAINT 组中的任何用户都将具有 SYSMAINT 权限。要从 xinzhuang 那里撤销 SYSMAINT 特权，只需从 MAINT 组中删除它，或者将 SYSMAINT_GROUP 参数的值改为另一个不包含它的组。在后一种情况下，如果 MAINT 组的其他成员不是新组的成员，那么它们的 SYSMAINT 权限也会被撤销。

实例级权限参数除了使用命令行修改外，也可通过控制中心进行修改。

获得 SYSADM 权限

DB2 中的 SYSADM 权限就像是 UNIX 上的 root 权限或 Windows 上的 Administrator 权限。对一个 DB2 实例拥有 SYSADM 权限的用户能够对这个实例，这个实例中的任何数据库，以及这些数据库中的任何对象发出任何 DB2 命令。他们还能够访问数据库中的数据，以及对其他用户授予或撤销特权或权限。只允许 SYSADM 用户更新 DBM CFG 文件。SYSADM 权限由 DBM CFG 文件中的 SYSADM_GROUP 参数控制。在 Windows 上，在创建实例时，这个参数设置为 Administrator(如果安全时启用操作系统安全性选项，该组为 db2admins)。但是，如果发出命令 db2 get dbm cfg，那么它看起来是空的。在 UNIX 上，

它设置为创建这个实例的用户的主组。

因为只允许 SYSADM 用户更新 DBM CFG 文件，所以只有他们能够向其他组授予任何 SYS*权限。以下示例演示如何向 *db2grp1* 组授予 SYSADM 权限：

```
db2 update dbm cfg using SYSADM_GROUP db2grp1
```

请记住，这一修改直到实例停止并重新启动之后才会生效。还要记住，如果您当前不是作为 *db2grp1* 组的成员登录的，那么就无权重新启动实例！您必须注销并用正确的组中的 ID 重新登录，或者将自己当前的 ID 添加进 *db2grp1* 组中。

获得 SYSCTRL 权限

拥有 SYSCTRL 权限的用户可以在实例中执行所有管理和维护命令。但是，与 SYSADM 用户不同，他们不能访问数据库中的任何数据，除非他们被授予了访问数据所需的特权。SYSCTRL 用户可以对实例中的任何数据库执行的命令示例如下所示：

- db2start/db2stop
- db2 create/drop database
- db2 create/drop tablespace
- db2 backup/restore/rollforward database
- db2 runstats(针对任何表)
- db2 update db cfg for database dbname

拥有 SYSADM 权限的用户可以使用以下命令将 SYSCTRL 分配给一个组：

```
db2 update dbm cfg using SYSCTRL_GROUP group name
```

获得 SYSMAINT 权限

拥有 SYSMAINT 权限的用户可以发出的命令是拥有 SYSCTRL 权限的用户可以发出的命令的子集。SYSMAINT 用户只能执行与维护相关的任务，比如：

- db2start/db2stop
- db2 backup/restore/rollforward database
- db2 runstats(针对任何表)
- db2 update db cfg for database dbname

注意，拥有 SYSMAINT 权限的用户不能创建或删除数据库或表空间。他们也不能访问数据库中的任何数据，除非他们被显式地授予访问数据所需的特权。

如果您拥有 SYSADM 权限，那么可以使用以下命令将 SYSMAINT 权限分配给一个组：

```
db2 update dbm cfg using SYSMAINT_GROUP group name
```

使用控制中心修改实例级权限

要使用控制中心修改实例级权限参数，应该打开控制中心，展开**所有系统**文件夹，展开目标系统，展开**实例**文件夹，右击目标实例(在这个例子中是 **DB2**)，并选择**配置参数**项(见图 13-9 所示)。

滚动参数列表(图 13-10 所示)并找到相关的权限级别参数。单击参数值旁边的按钮来改变它的值。在图 13-10 所示的例子中，将 SYSMAINT_GROUP 参数的值改为您指定的组名。



图 13-9　打开控制中心中的**配置参数**对话框

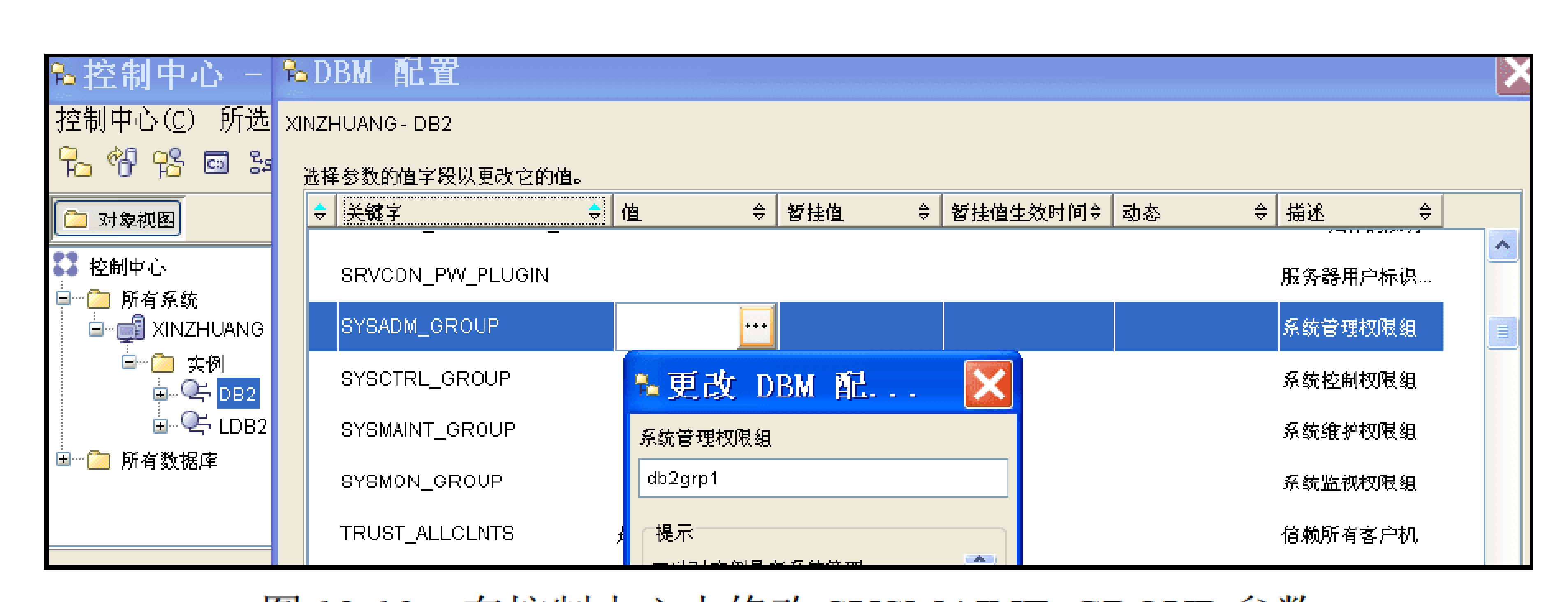


图 13-10　在控制中心中修改 SYSMAINT_GROUP 参数

必须停止并重新启动实例，对参数的修改才会生效。在控制中心中，再次右击目标实例，并选择**停止**项。如果提示对是否停止实例进行确认，那么单击 **OK** 按钮。再次右击目标实例，并选择**启动**项。然后就可以检查参数是否已经生效了。

在 Windows 上的默认 DB2 安装中，这些实例级权限参数默认为 NULL。这意味着任何属于本地 Administrators 组的用户账号自动继承这些权限。因此，强烈建议明确地将这些参数的值改为特定的组名，以避免意外的/未授权的访问。在 Linux 和 UNIX 安装上，这不是个大问题，因为 NULL 值默认表示实例拥有者的主组，而这个组在安装之后默认情况下只包含实例拥有者的用户 ID。但是，明确地设置这些参数仍然是好的做法。

13.3.3 授予/撤销数据库级权限

数据库级权限包括 DBADM、CONNECT、CREATETAB、SECADM 和 LOAD。

获得 DBADM 权限

DBADM 权限是一个数据库级权限，而不是实例级权限。DBADM 用户对一个数据库有几乎完全的控制能力。DBADM 用户不能执行某些维护或管理任务，比如：

- drop database
- drop/create tablespace
- backup/restore database
- update db cfg for database *db name*

但是，他们可以执行以下任务：

- db2 create/drop table
- db2 grant/revoke(任何特权)
- db2 runstats(任何表)

DBADM 用户还被自动地授予对数据库对象及其内容的所有特权。因为 DBADM 权限是一个数据库级权限，所以它可以被分配给用户和用户组。以下命令演示了授予 DBADM 权限的不同方法。

```
db2 create database test
```

这个命令将数据库 *test* 上的 DBADM 权限隐式地授予发出此命令的用户。

```
db2 connect to sample
db2 grant dbadm on database to user xinzhuang
```

这个命令只能由 SYSADM 用户发出，它向用户 xinzhuang 授予 Sample 数据库上的 DBADM 权限。注意，在授予 DBADM 权限之前，发出这个命令的用户必须连接到示例数据库。

```
db2 grant dbadm on database to group db2grp1
```

这个命令将 DBADM 权限授予 *db2grp1* 组中的每个用户。同样，只有 SYSADM 用户能够发出这个命令。

获得 LOAD 权限

LOAD 权限是一个数据库级权限，所以它可以被分配给用户和用户组。顾名思义，LOAD 权限允许用户对表发出 LOAD 命令。当用大量数据填充表时，LOAD 命令通常用来替代插入或导入命令，它的速度更快。根据执行的 LOAD 操作类型的不同，有时候仅仅

拥有 LOAD 权限可能还不够。可能还需要表上的特定特权。

拥有 LOAD 权限的用户可以运行以下命令：

- db2 quiesce tablespaces for table
- db2 list tablespaces
- db2 runstats(任何表)
- db2 load insert(必须有表上的插入特权)
- db2 load restart/terminate after load insert(必须有表上的插入特权)
- db2 load replace(必须有表上的插入和删除特权)
- db2 load restart/terminate after load replace(必须有表上的插入和删除特权)

只有拥有 SYSADM 或 DBADM 权限的用户能够对用户或用户组授予或撤销 LOAD 权限。以下示例演示了 LOAD 权限如何允许我们的用户使用 LOAD 命令将数据装载进 *sales* 表中(假设已经发出了命令 db2 connect to sample)：

```
db2 grant load on database to user xinzhuang
db2 grant insert on table sales to user xinzhuang
```

有了 LOAD 权限和插入特权，xinzhuang 就可以对 sales 表发出 LOAD INSERT 或 LOAD RESTART，或者在 LOAD INSERT 之后发出 TERMINATE。看下面例子：

```
db2 grant load on database to group grp1
db2 grant delete on table sales to group grp1
db2 grant insert on table sales to group grp1
```

有了 LOAD 权限以及删除和插入特权，*grp1* 的任何成员就可以对 sales 表发出 LOAD REPLACE 或 LOAD RESTART，或者在 LOAD REPLACE 之后发出 TERMINATE。

13.4 特权

13.4.1 特权层次结构

实例权限级别这种机制用于将一组预先定义的管理权限授予一组用户账号，而特权会明确分配给单独用户或组，允许他们在数据库对象上执行特定操作(例如创建和删除表)。特权给予用户通过特定方式访问数据库对象的权力，特权严格地定义了用户可以执行的任务。例如，用户可能具有读一个表的数据的特权，但是不能更新这些数据。图 13-11 给出了不同数据库对象的特权摘要。特权大体上分成两类：数据库特权(针对数据库中所有对象)和对象级特权(与特定对象相关联)。

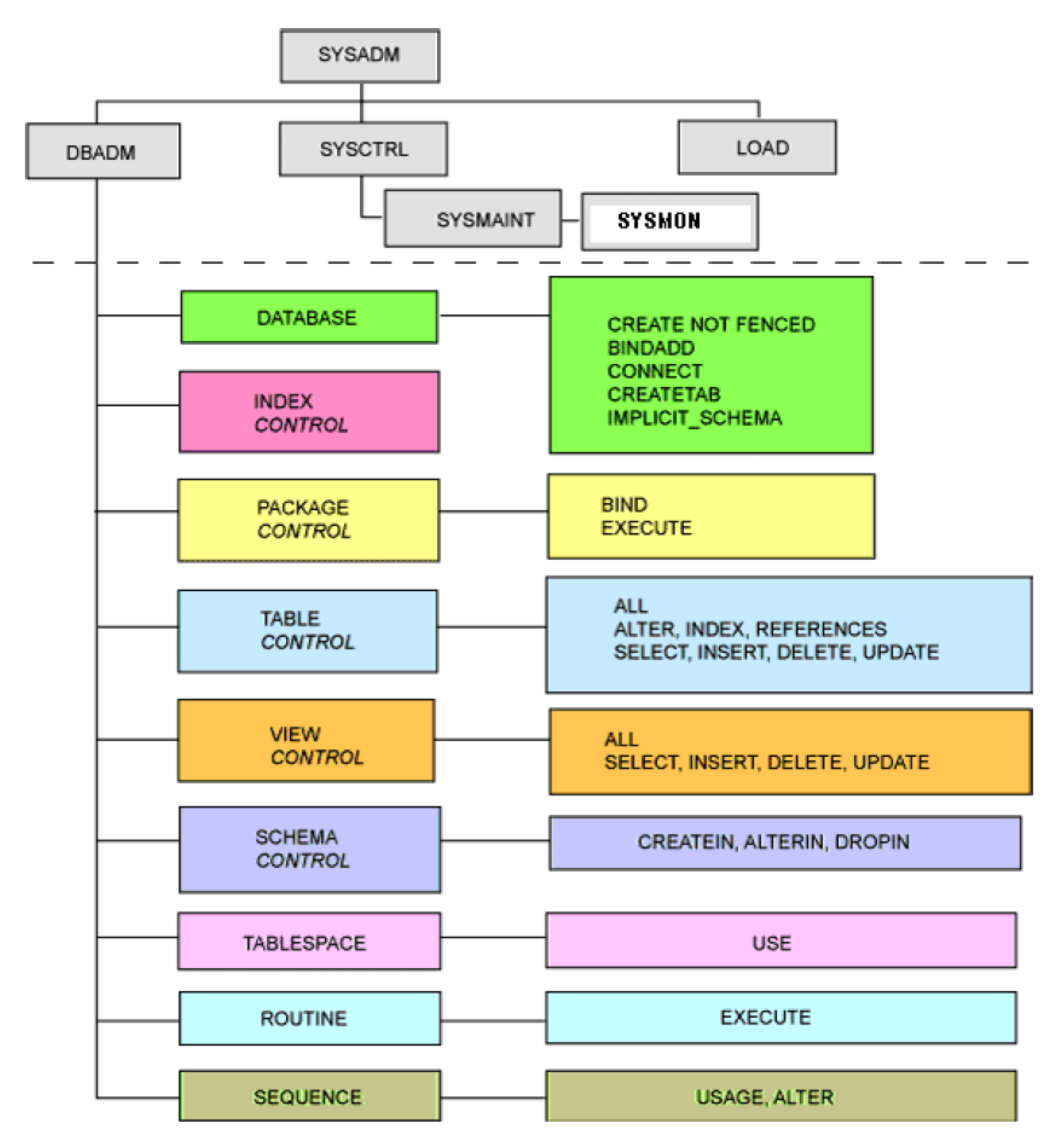


图 13-11 DB2 权限和特权的层次结构

图 13-10 显示了 DB2 中不同的权限和特权级别。范围涵盖了表、模式、存储过程等不同对象上的特权。图 13-10 顶部虚线上方的灰色部分显示了前一节描述的实例权限级别。注意 SYSADM 和 DBADM 权限自动获得虚线下面对于某个数据库的所有权限和特权。

表 13-3 总结了对用户或用户组可以授予和撤销的数据库权限类型。只有具有 SYSADM 或 DBADM 权限的用户可以授予和撤销这些权限。

表 13-3 数据库级权限总结

数据库权限	说　明
CONNECT	允许用户连接数据库
BINDADD	允许用户在数据库中创建新的包
CREATETAB	允许用户在数据库中创建新的表

(续表)

数据库权限	说　　明
CREATE_NOT_FENCED	允许用户注册定义为 NOT FENCED 的用户定义函数(UDF)或存储过程
IMPLICIT SCHEMA	允许用户在尚不存在的模式中创建对象(它自动地创建模式)*
QUIESCE_CONNECT	允许用户连接处于 quiesced 状态的数据库
CREATE_EXTERNAL_ROUTINE	允许用户注册外部例程(用 C 和 Java 等外部语言编写的例程)

*SYSIBM 成为隐式创建的模式的所有者，PUBLIC 组被授予在这个模式中创建对象的特权。

表 13-4 总结了对用户或用户组可以授予和撤销的唯一一种表空间特权(USE)。USE 特权不能对 SYSCATSPACE 或任何系统临时表空间使用。

表 13-4　表空间特权总结

表空间特权	说　　明
USE	允许用户在指定的表空间中创建表

表 13-5 总结了对用户或用户组可以授予和撤销的模式特权类型。

表 13-5　模式特权总结

模 式 特 权	说　　明
CREATEIN	允许用户在模式中创建对象
ALTERIN	允许用户在模式中修改对象
DROPIN	允许用户从模式中删除对象

表 13-6 总结了对用户或用户组可以授予和撤销的表/视图特权。

表 13-6　表和视图特权总结

表/视图特权	说　　明
CONTROL	授予用户在表和视图上的所有特权，以及将这些特权(除了 CONTROL)授予别人
ALTER	允许用户在表中添加列，在表和它的列上添加或修改注释，添加主键或唯一约束，以及创建或删除表检查约束

(续表)

表/视图特权	说　明
DELETE	允许用户从表或视图中删除行
INDEX	允许用户在表上创建索引
INSERT	允许用户在表或视图中插入数据
REFERENCES	允许用户创建和删除外键，这需要指定关系中的父表
SELECT	允许用户从表或视图中检索行，在表上创建视图，以及运行 EXPORT 实用程序
UPDATE	允许用户修改表、视图或者表或视图中某些列中的数据；用户可以只在特定列上具有这种特权

表 13-7 总结了对用户或用户组可以授予和撤销的唯一一种索引特权(CONTROL)。

表 13-7　索引特权总结

索 引 特 权	说　明
CONTROL	允许用户删除索引

表 13-8 总结了对于用户或用户组可以授予和撤销的包特权类型。

表 13-8　包特权总结

包　特　权	说　明
CONTROL	允许用户重新绑定、删除或执行包，以及将这些特权(除了 CONTROL)授予别人
BIND	允许用户重新绑定现有的包
EXECUTE	允许用户执行包

表 13-9 总结了对用户或用户组可以授予和撤销的唯一一种例程特权(EXECUTE)。

表 13-9　例程特权总结

例程特权	说　明
EXECUTE	允许用户调用例程，从例程创建函数(只应用于函数)，以及在任何 DDL 语句(比如 CREATE VIEW、CREATE TRIGGER 或定义约束时)中引用例程

表 13-10 总结了对用户或用户组可以授予和撤销的序列特权类型。

表 13-10　序列特权总结

序 列 特 权	说　　明
USAGE	允许用户对序列使用 NEXTVAL 和 PREVVAL 表达式
ALTER	允许用户使用 ALTER SEQUENCE 语句修改序列属性

13.4.2　授予特权

与实例级权限相似，可以使用命令语法或控制中心授予和撤销特权。要想授予或撤销特权，必须有数据库连接。图 13-12 显示了表和视图特权的 GRANT 语句的语法。其他数据库对象的 GRANT 语句语法与此相似。

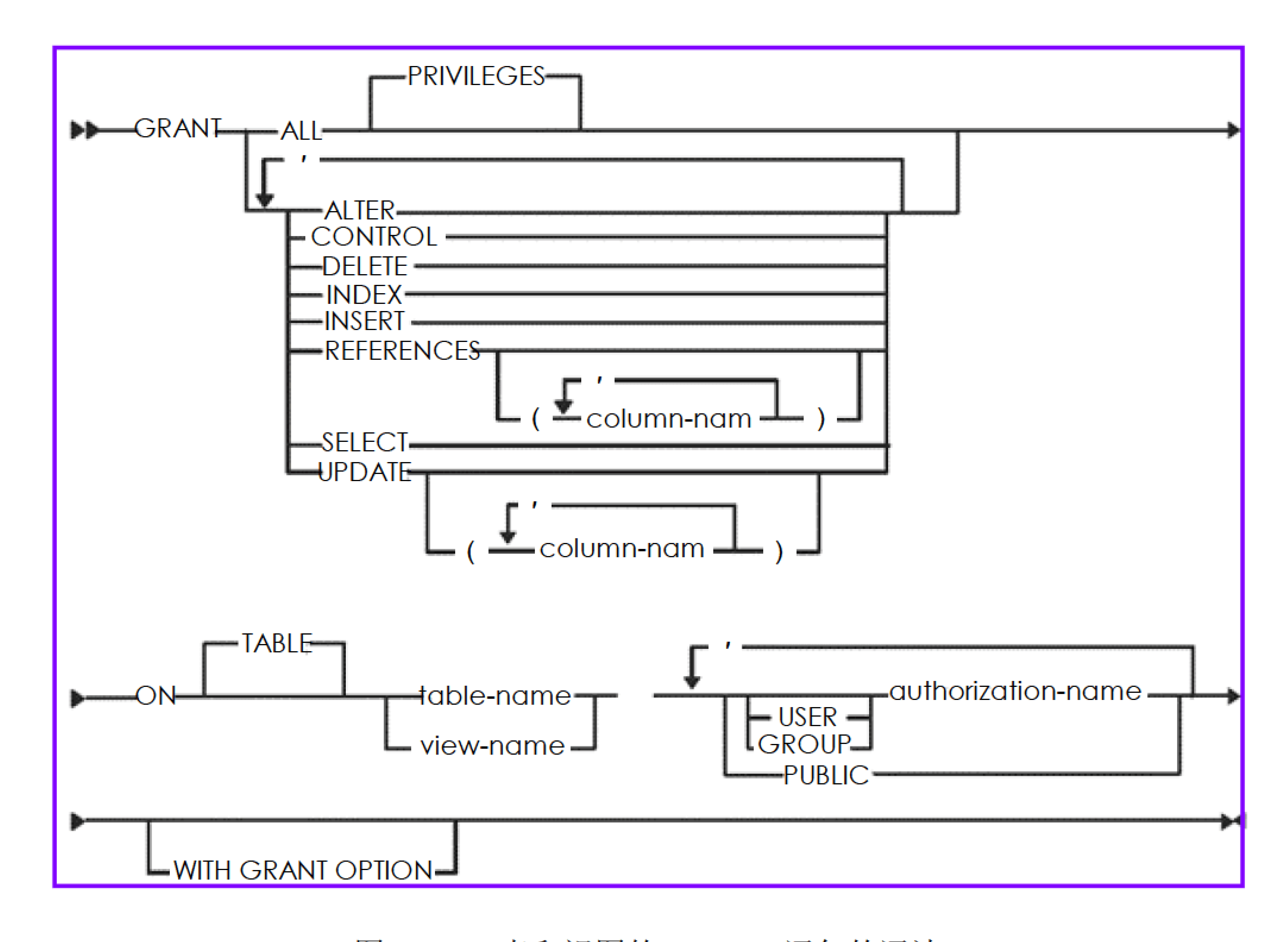


图 13-12　表和视图的 GRANT 语句的语法

例如，要使用 GRANT 语句向用户 *nxz* 授予 *ACCOUNT* 表的 INSERT 特权，执行以下语句：

```
GRANT INSERT ON TABLE account TO USER nxz
```

表 13-10 总结了对用户或用户组可以授予和撤销的序列特权类型。

表 13-10　序列特权总结

序 列 特 权	说　明
USAGE	允许用户对序列使用 NEXTVAL 和 PREVVAL 表达式
ALTER	允许用户使用 ALTER SEQUENCE 语句修改序列属性

13.4.2　授予特权

与实例级权限相似，可以使用命令语法或控制中心授予和撤销特权。要想授予或撤销特权，必须有数据库连接。图 13-12 显示了表和视图特权的 GRANT 语句的语法。其他数据库对象的 GRANT 语句语法与此相似。

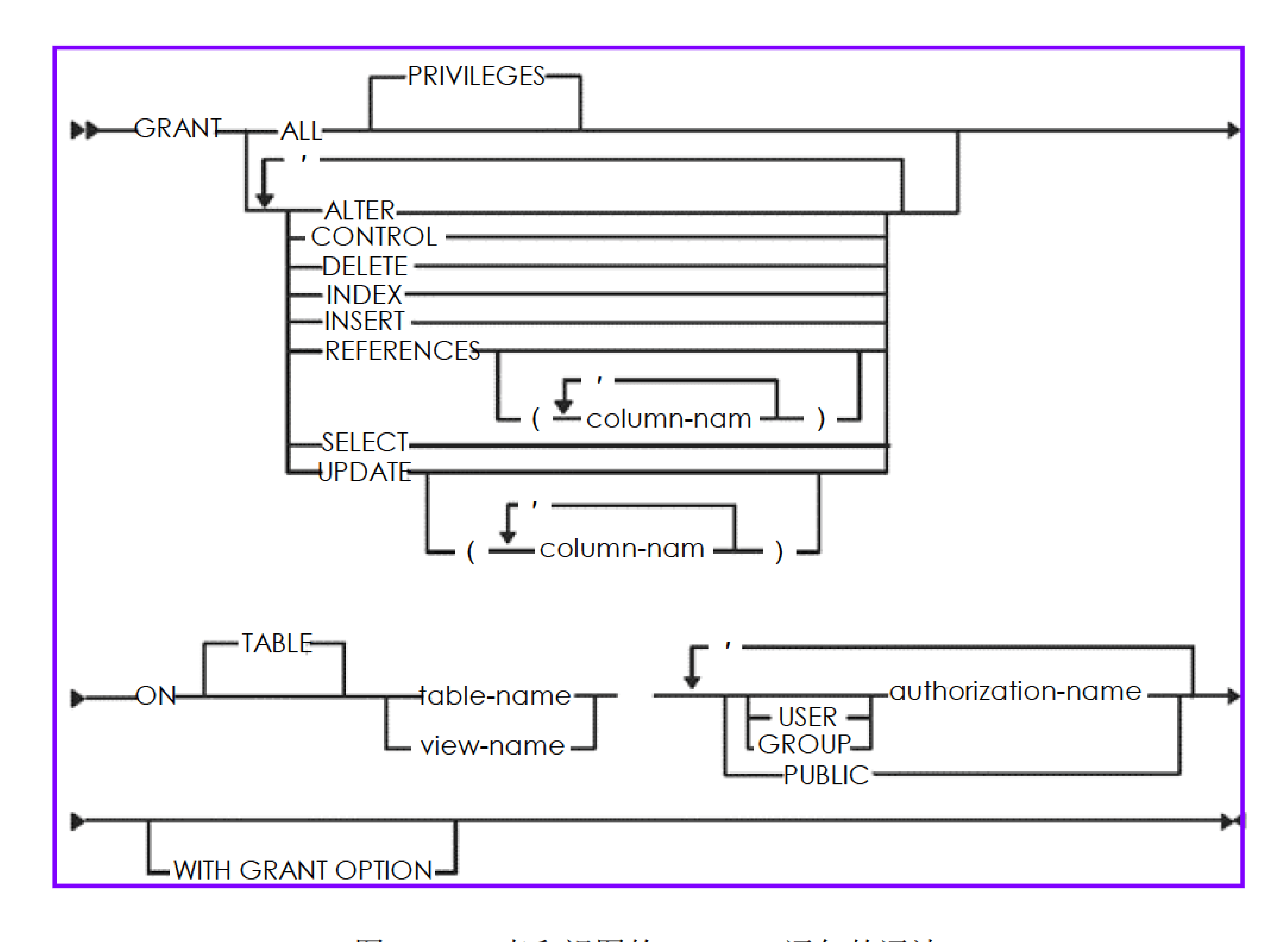


图 13-12　表和视图的 GRANT 语句的语法

例如，要使用 GRANT 语句向用户 *nxz* 授予 *ACCOUNT* 表的 INSERT 特权，执行以下语句：

```
GRANT INSERT ON TABLE account TO USER nxz
```

要向 *grp1* 组授予 *CUSTOMER* 表的 SELECT 特权，执行以下语句：

```
GRANT SELECT ON TABLE customer TO GROUP grp1
```

在向用户或组授予权限和特权时必须小心，因为 DB2 允许将这些特权授予不存在的账号。如果以后创建了同名的账号，那么这个账号会自动获得以前授予的所有权限和特权。

使用控制中心授权

还可以使用控制中心授予特权，办法是展开**所有数据库**文件夹，展开目标数据库，展开包含感兴趣的数据库对象的文件夹，右击这个对象，并选择**特权** 项。在图 13-13 中，展开 SAMPLE 数据库中的**表**文件夹，右击 EMPLOYEE 表并选择**特权**项。

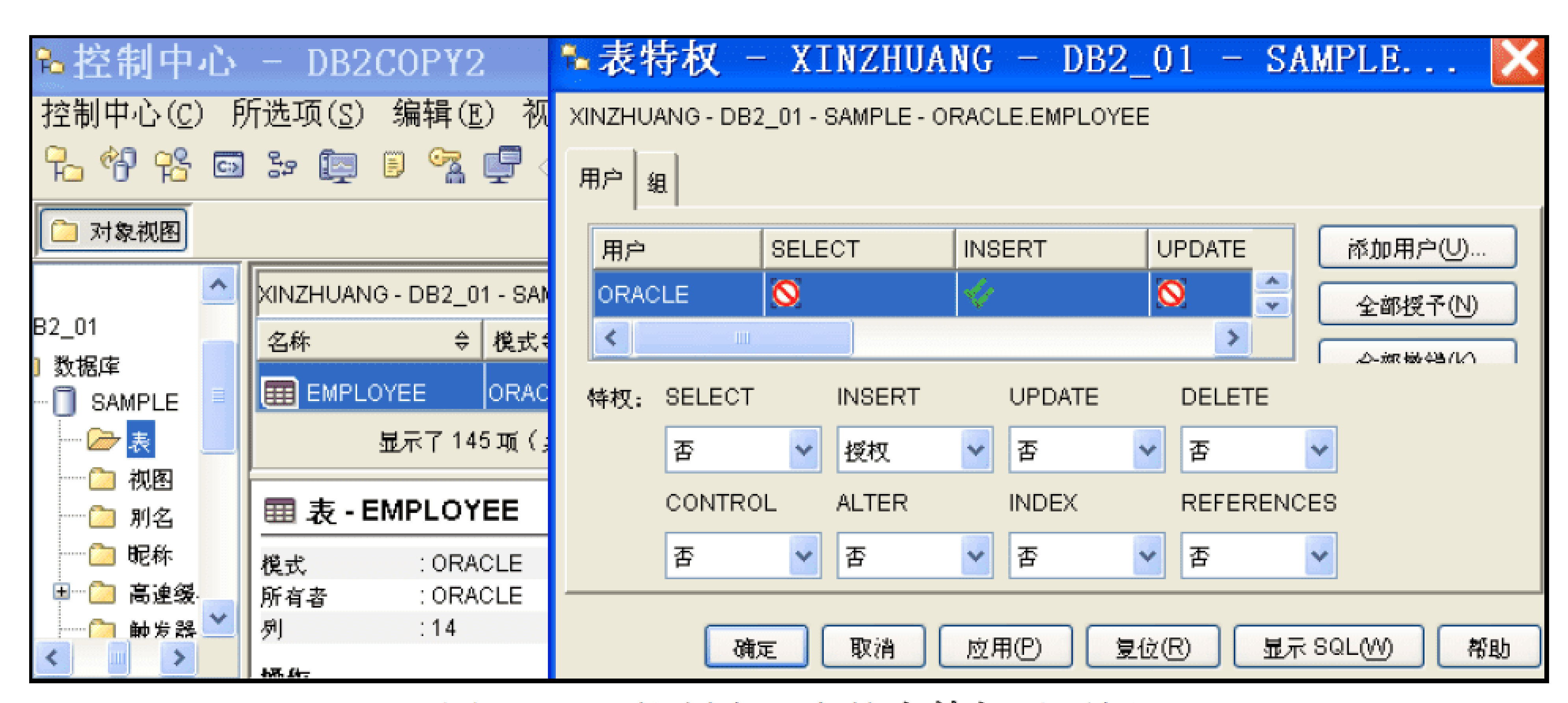


图 13-13　控制中心中的**表特权**对话框

在**表特权**对话框中，根据是要向用户还是组授予特权，选择**用户**或**组**。如果用户/组不在列表中，那么单击**添加用户**或**添加组**按钮添加用户或组。通过单击适当特权的下拉框并选择 **Yes**、**No** 或 **Grant**，从而指定应该向用户或组授予哪些特权。选择 **Yes** 意味着应该授予特权，选择 **No** 意味着应该不授予特权，选择 **Grant** 意味着应该授予此特权和向其他用户/组授予此特权的特权。单击 **Grant All** 按钮向指定用户或组授予所有可用的特权。单击 **Revoke All** 按钮从指定用户或组撤销所有可用的特权。

在图 13-12 中可以看到，用户 *ORACLE* 只被授予 EMPLOYEE 表上的 INSERT 特权，这意味着 ORACLE 只能在这个表中插入数据，不能读或更新它。当然，这里假设 *ORACLE* 不是具有这些特权或 SYSADM/DBADM 权限的组的成员。

向用户与组授予特权

使上面的例子或 GRANT 语句语法图中可以看出，可以分别使用 TO USER 或 TO GROUP 子句指定是向用户还是向组授予特权。如果没有指定这两个子句之一，且指定的

名称在操作系统中只定义为组，那么 DB2 会假设把特权授予 GROUP；如果指定的名称在操作系统中只定义为用户，或者没有定义，那么 DB2 会假设把特权授予 USER。如果指定的名称在操作系统中同时定义为用户和组，那么将返回一个错误。作为最佳实践，我们建议在 GRANT 语句中总是包含 TO USER 或 TO GROUP 子句，以避免任何二义性。

PUBLIC 组

DB2 在内部使用一个伪组 PUBLIC，可以对它授予和撤销特权。PUBLIC 实际上并不是外部安全设施中定义的一个组，而是一种向成功经过身份认证的用户分配特权的方式。可以对 PUBLIC 组授予和撤销特权，就像对其他组一样。例如，要从 PUBLIC 组撤销 IMPLICIT_SCHEMA 权限，可以发出以下语句：

```
REVOKE IMPLICIT_SCHEMA ON DATABASE FROM PUBLIC
```

重要的是，要理解向 PUBLIC 组授予特权的安全影响。任何提供了有效用户 ID 和密码的用户都能够执行 PUBLIC 组有权执行的操作。

WITH GRANT OPTION

许多数据库对象特权还允许在 GRANT 语句中包含 WITH GRANT OPTION 子句。这使您能够将一种特权授予用户或组，同时使用户或组的成员能够将同一特权授予别的用户或组。例如，以下语句将 *ACCT* 模式上的 ALTERIN、CREATEIN 和 DROPIN 特权授予组 *G1*，同时允许组 *G1* 的成员将这些特权授予别的用户或组：

```
GRANT ALTERIN, CREATEIN, DROPIN ON SCHEMA ACCT TO GROUP G1 WITH GRANT OPTION
```

WITH GRANT OPTION 只能用于包、例程、模式、表、视图和表空间的 GRANT 语句。

注意：

不能使用 WITH GRANT OPTION 子句向别的用户或组授予对象的 CONTROL 特权。这种特权必须专门授予用户或组，并只能由具有 SYSADM 或 DBADM 权限的用户执行。

13.4.3 撤销特权

REVOKE 语句用于撤销以前授予的特权。图 13-14 显示了表和视图的 REVOKE 语句的语法。同样，其他数据库对象的 REVOKE 语法与此相似。

例如，要从用户 *JEN* 撤销 *STAFF* 表上的 ALTER 特权，可以发出以下语句：

```
REVOKE ALTER ON TABLE staff FROM USER jen
```

要从 *JEN* 撤销 *STAFF* 表上的所有特权，可以发出以下语句：

```
REVOKE ALL PRIVILEGES ON TABLE staff FROM USER jen
```

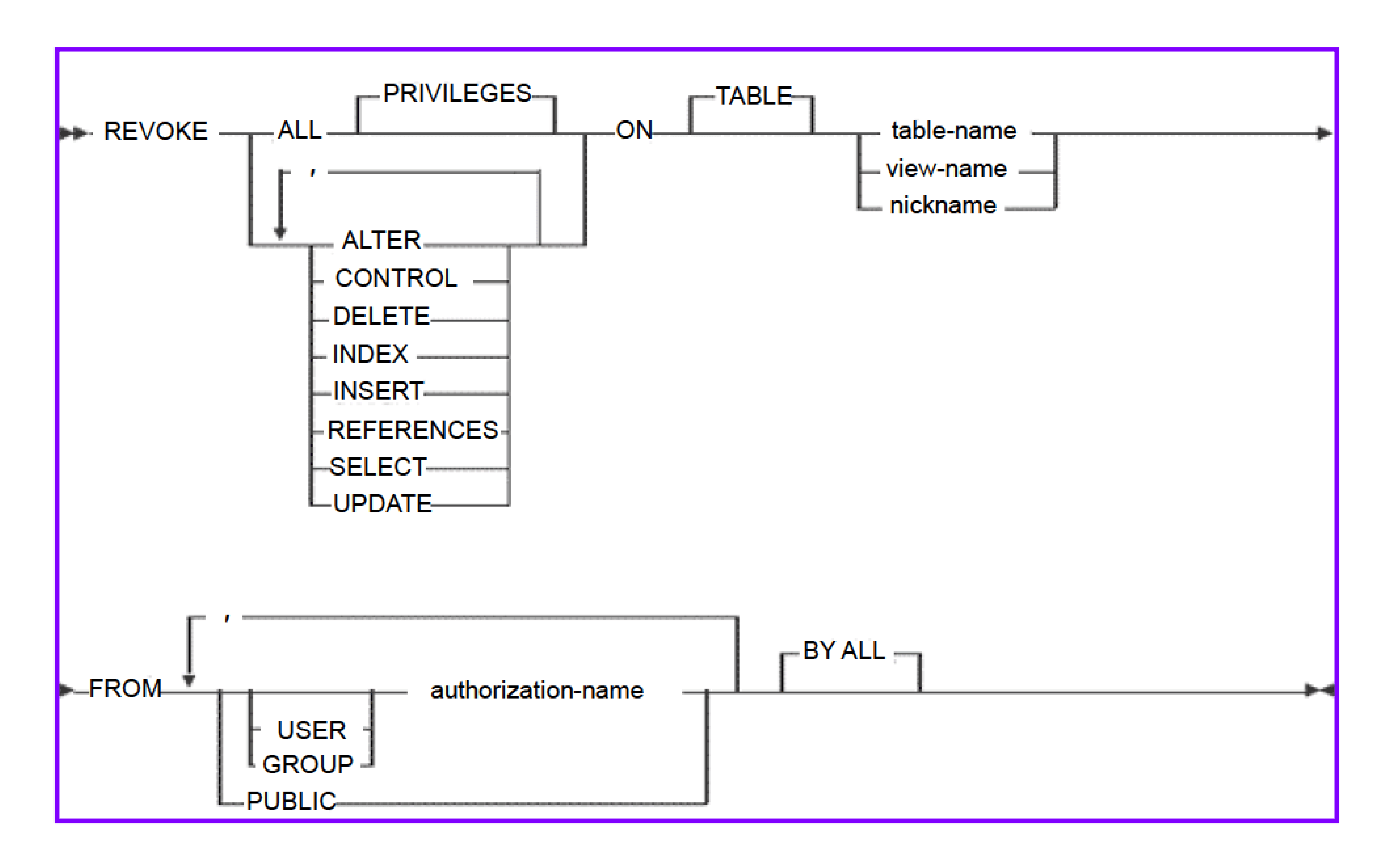


图 13-14　表和视图的 REVOKE 语句的语法

还可以使用控制中心撤销特权，操作方式与授予特权相似。只需重新打开对象特权对话框，如前面的图 13-12 所示。要撤销一种特权，只需将此特权的下拉列表改为 **NO**，或者单击 **Revoke All** 按钮撤销与此对象相关的所有特权。

要撤销数据库对象上的特权，必须具有 DBADM 权限、SYSADM 权限或此对象上的 CONTROL 特权。注意，拥有 WITH GRANT OPTION 特权并不足以撤销这一特权。要从另一个用户撤销 CONTROL 特权，必须具有 SYSADM 或 DBADM 权限。

从用户或组撤销特权是撤销其他任何账号授予他的特权。但是从用户或组撤销特权并不撤销这个用户/组授予别的账号的同一特权。例如，假设用户 *BEN* 将 SELECT WITH GRANT OPTION 授予用户 *RICK*，然后 *RICK* 将 SELECT 授予用户 *RAVI* 和 *CHRIS*。如果以后 *BEN* 从 *RICK* 撤销了 SELECT 特权，那么 *RAVI* 和 *CHRIS* 仍然拥有 SELECT 特权。但是如果 BEN 的权限没有了，那么 RICK、RAVI 和 CHRIS 等用户的 select 特权也都没有了。

从组中的一个成员撤销特定特权

还有一种情况：希望将一种特权授予一个组，然后只从这个组中的一个成员撤销 CONTROL 这一特权。但是，不能撤销并未明确授予的特权。在这种情况下，有两种办法：

- 可以从这个组中删除这个成员；或者创建一个只包含组中其他成员的新组，并将特权授予这个新组。

- 可以从这个组撤销特权，然后向组的各个成员分别授予特权。

授予和撤销数据库权限

也可使用 GRANT 语句将数据库级权限(比如 DBADM、LOAD 和 CREATETAB)授予用户或组。例如，以下语句将 DBADM 权限授予用户 *SALLY*：

```
GRANT DBADM ON DATABASE TO USER sally
```

以下语句将 LOAD 权限授予组 *MAINT*：

```
GRANT LOAD ON DATABASE TO GROUP maint
```

如果 LOAD 操作定义为 REPLACE，那么具有 LOAD 权限的用户还需要 INSERT(为了将数据装载到表中)和 DELETE 特权。

要撤销数据库级权限，使用 REVOKE 语句。例如，要从组 MAINT 撤销 LOAD 权限，可以发出以下语句：

```
REVOKE LOAD ON DATABASE FROM GROUP maint
```

注意：

要撤销 DBADM 权限，必须具有 SYSADM 权限。

13.4.4 显式特权/隐式特权/间接特权

显式特权

可以使用 GRANT 和 REVOKE 命令显式地对用户或组授予或撤销特权。我们来看看如何在各种对象上使用这些命令。

以拥有 Administrator 权限的用户登录 Windows，打开两个 DB2 命令窗口。在这两个窗口中，确保将 db2instance 变量设置为 DB2。

在第一个窗口中发出以下命令：

```
db2 connect to sample
```

现在，在第二个窗口中发出以下命令：

```
db2 connect to sample user test1 using password
```

请记住，第一个窗口中的命令是由一个拥有 SYSADM 权限的用户发出的。第二个窗口中的命令是由 *test1* 发出的，这个用户对示例数据库没有特殊的权限或特权。注意，与示例数据库中的表相关联的模式名是发出 db2sampl 命令的用户的名称。在这些示例中，这个

用户是 *GMILNE*。

现在，在第二个窗口中发出以下命令：

```
db2 select * from gmilne.org
```

应该会看到以下响应：

```
SQL0551N  "TEST1" does not have the privilege to perform operation "SELECT" on object "GMILNE.ORG".
```

为了纠正这种状况，在第一个窗口中发出以下命令：

```
db2 grant select on table gmilne.org to user test1
```

现在，前面的命令就会成功！接下来，在第二个窗口中发出一个更复杂的命令：

```
db2 insert into gmilne.org values(100, 'NXZ', 1, 'NXZ', 'NXZ')
```

同样会看到错误消息：

```
SQL0551N  "TEST1" does not have the privilege to perform operation  "INSERT" on object "GMILNE.ORG"
```

所以，在第一个窗口中输入以下命令：

```
db2 grant insert on table gmilne.org to group db2grp1
```

原来失败的 INSERT 命令现在应该会成功完成，因为 *test1* 是 *db2grp1* 组的成员。

现在，在第二个窗口中输入以下命令：

```
db2 drop table gmilne.emp_photo
```

同样会看到错误消息：

```
SQL0551N  "TEST1" does not have the privilege to perform operation "DROP TABLE" on object "GMILNE.EMP_PHOTO".
```

所以，我们要授予这个特权。在第一个窗口中输入以下命令：

```
db2 grant dropin on schema gmilne to all
```

DROP TABLE 命令现在应该会成功完成。

既然已经完成了示例，就可以撤销刚才授予的特权。在第一个窗口中发出以下命令：

```
db2 revoke select on table gmilne.org from user test1
db2 revoke insert on table gmilne.org from group db2grp1
```

```
db2 revoke dropin on schema gmilne from all
```

注意，从组中撤销特权不一定会从这个组的所有成员撤销该特权。例如，以下命令可以用来从 *db2grp1* 撤销对 gmilne.org 表的所有特权(CONTROL 除外)：

```
db2 revoke all on table gmilne.org from group db2grp1
```

但是，*test1* 用户(*db2grp1* 的成员)仍然拥有对这个表的选择特权，因为 test1 是被直接授予这个特权的。

隐式特权

当发出某些命令时，DB2 可能会自动地授予特权，而不需要像前面看到的那样发出显式的 GRANT 语句。表 13-11 总结了会导致数据库管理程序隐式地授予特权的一些命令。注意，当删除创建的对象时，这些特性会隐式地撤销。但是，当显式地撤销更高级的特权时，不会撤销这些隐式特权。

表 13-11　会导致数据库管理程序隐式授予特权的一些命令

发出的命令	授予的特权	被授予特权的用户
CREATE TABLE mytable	mytable 上的 CONTROL	发出命令的用户
CREATE SCHEMA myschema	myschema 上的 CREATEIN、ALTERIN 和 DROPIN，以及将这些特权授予其他用户的能力	发出命令的用户
CREATE VIEW myview	myview 上的 CONTROL(只有在用户拥有 myview 定义中引用的所有表和视图上的 CONTROL 特权的情况下)	发出命令的用户
CREATE DATABASE mydb	mydb 的系统编目表上的 SELECT，mydb 上的 IMPLICIT_SCHEMA *	PUBLIC**

*表示当用户创建数据库时，隐式地授予这个用户这个数据库上的 DBADM 权限。获得 DBADM 权限就会隐式地授予 CONNECT、CREATETAB、BINDADD、IMPLICIT_SCHEMA 和 CREATE_NOT_FENCED 特权。即使撤销了 DBADM 权限，这个用户仍然会保留这些特权。

**表示 PUBLIC 是一个特殊的 DB2 组，其中包括特定数据库的所有用户。与前面讨论过的其他组不同，PUBLIC 不必在操作系统级进行定义。在默认情况下，会向 PUBLIC 授予一些特权。例如，这个组自动接受数据库上的 CONNECT 特权和编目表上的 SELECT 特权。可以对 PUBLIC 组发出 GRANT 和 REVOKE 命令，比如：

```
db2 grant select on table sysibm.systables to public
db2 revoke select on table sysibm.systables from public
```

在某些情况下，当用户创建一个数据库对象(比如表或包)时，或者授予 DBADM 权限级别时，数据库管理器会隐式地将某些特权授予用户。重要的是，要了解授予了哪些隐式特权以及这些隐式特权的安全影响。表 13-12 总结了授予隐式特权的情况。

表 13-12　对于不同操作授予隐式特权的总结

操　　作	向执行此操作的用户授予的隐式特权
创建新数据库	将 DBADM 权限以及 BINDADD、CONNECT、CREATETAB、CREATE_EXTERNAL_ROUTINE、CREATE_NOT_FENCED_ROUTINE、IMPLICIT_SCHEMA、LOAD 和 QUIESCE_CONNECT 权限授予创建者(SYSADM 或 SYSCTRL) 将 BINDADD、CREATETAB、CONNECT 和 IMPLICIT_SCHEMA 授予 PUBLIC 将每个成功绑定的实用程序上的 BIND 和 EXECUTE 特权授予 PUBLIC 将系统编目表和视图上的 SELECT 授予 PUBLIC 将 USERSPACE1 表空间上的 USE 特权授予 PUBLIC 将 SYSFUN 模式中的所有函数上的 EXECUTE WITH GRANT 特权授予 PUBLIC 将 SYSIBM 模式中的所有过程上的 EXECUTE 特权授予 PUBLIC
授予 DBADM 权限	授予 BINDADD、CONNECT、CREATETAB、CREATE_EXTERNAL_ROUTINE、CREATE_NOT_FENCED_ROUTINE、IMPLICIT_SCHEMA、LOAD 和 QUIESCE_CONNECT
模式	在显式创建时，将 CREATEIN、ALTERIN、DROPIN 授予创建这个模式的用户 在隐式创建时，将 CREATEIN 授予 PUBLIC
创建对象(表、索引、包)	将 CONTROL 授予对象创建者
创建视图	只有在用户拥有视图定义中引用的所有表、视图和别名的 CONTROL 特权时，授予 CONTROL 特权

例如，假设最初将 DBADM 权限授予用户 *PAUL*，以后决定撤销此权限。要从 *PAUL* 撤销 DBADM 权限，可以使用以下语句：

```
REVOKE DBADM ON DATABASE FROM USER paul
```

在执行这个命令之后，*PAUL* 不再拥有 DBADM 权限；但是，他仍然拥有数据库上的 GRANT、BINDADD、CONNECT、CREATETAB、CREATE_EXTERNAL_ROUTINE、CREATE_NOT_FENCED_ROUTINE、IMPLICIT_SCHEMA、LOAD 和 QUIESCE_CONNECT 权限，这些权限是原来将 DBADM 权限授予 *PAUL* 时隐式授予的。需要从 *PAUL* 显式地撤

销这些权限。

一种好的安全实践是，在创建新数据库之后，显式地撤销隐式授予 PUBLIC 的许多特权。这可以保护系统，确保只有应该访问数据库的用户才能这么做。

间接特权

当数据库管理器执行*包*时，可以间接获得特权。包中包含一个或多个 SQL 语句，这些语句已经转换为 DB2 用来在内部执行它们的格式。换句话说，包中包含可执行格式的多个 SQL 语句。如果包中的所有语句都是静态的，那么用户只需要有包上的 EXECUTE 特权，就能够成功地执行包中的语句。

例如，假设 *db2package1* 执行以下静态的 SQL 语句：

```
db2 select * from org
db2 insert into test values(1, 2, 3)
```

在这种情况下，拥有 *db2package1* 上的 EXECUTE 特权的用户会间接地获得 org 表上的 SELECT 特权和 test 表上的 INSERT 特权。

13.4.5　静态和动态 SQL 特权考虑因素

如果在编译时一个 SQL 语句的语法是完全已知的，那么这个语句就称为静态 SQL 语句。反之就是动态 SQL 语句，其语法直到运行时才知道。

在静态和动态 SQL 之间，处理特权的方式有一些差异。这些差异之一是如何处理组成员关系。一般来说，对动态 SQL 和非数据库对象授权(比如实例级命令和实用程序)时，需要考虑组成员关系；而静态 SQL 在执行前就已经生成了固定的执行计划并且作为数据包存放，因此不需要考虑组成员关系。这种一般性规则的一个例外发生在特权被授予 PUBLIC 时；在这种情况下，在处理静态 SQL 时也需要考虑组成员关系。

另一个差异是实际进行授权的时间。如果用静态 SQL 语句编写一个程序，那么必须先在数据库中创建应用程序包(包含 SQL 语句的优化执行计划)，然后相关的程序才能执行程序包中包含的 SQL 语句。对静态 SQL 的授权发生在编译或绑定时。在运行时，用户只需要有包上的 EXECUTE 特权，就能够执行其中的语句。这意味着用户不能直接访问底层数据库对象。对底层数据库对象的访问只能通过包中的特定语句进行。对于动态 SQL 语句而言，授权针对每个语句进行。执行语句的用户必须具有适当的特权，才能在运行时执行语句，特权可以是明确授予他们的，也可以是通过组成员关系授予的。

例如，假设在一个嵌入式 SQL 应用程序中有以下的静态 SQL 语句：

```
EXEC SQL SELECT col INTO :hostvar FROM staff;
```

假设包含这个语句的应用程序文件已经预先编译了，产生了绑定文件 myapp.bnd。如果开发人员 *nxz* 希望将这个文件绑定到数据库(因此创建一个包)，那么她需要具有 *staff* 上的 SELECT 特权，才能成功执行 BIND 命令。这个规则有一个例外，即如果 PUBLIC 组已经被授予了这个特权，那么就不需要明确授予她这个特权。*nxz* 还需要 BINDADD 权限(如果这是一个新的包)或 BIND 特权(假设这是一个现有的包，她希望将它重新绑定到数据库，因为数据库统计值最近更新了)。

程序包也可能包含动态 SQL，在这种情况下，需要的特权取决于在对包进行预编译或绑定到数据库时 RECOMPILE/BIND 命令的 DYNAMICRULES 子句所指定的值。如果包是使用 DYNAMICRULES RUN(默认值)绑定的，那么要想使用其中的动态 SQL，运行动态 SQL 应用程序的用户必须具有发出每个 SQL 请求所需的特权，以及包上的 EXECUTE 特权。特权可以被授予用户的 ID、用户所在的任何组或 PUBLIC。

如果应用程序是使用 DYNAMICRULES BIND 选项绑定的，那么 DB2 将包所有者的用户 ID 与应用程序包关联起来。这使运行这个应用程序的任何用户能够继承与包所有者的用户 ID 相关联的特权。

例如，假设以上例子中的应用程序文件也包含动态 SQL，如例 13-1 所示。

例 13-1 一个嵌入式 SQL 应用程序中的动态 SQL。

```
EXEC SQL BEGIN DECLARE SECTION;
  char hostVarStmt[50];
  short hostVarDeptnum;
EXEC SQL END DECLARE SECTION;
strcpy(hostVarStmt, "DELETE FROM org WHERE deptnum = ?");
EXEC SQL PREPARE Stmt1 FROM :hostVarStmt;
hostVarDeptnum = 15;
EXEC SQL EXECUTE Stmt1 USING :hostVarDeptnum;
```

如果 *nxz* 希望将同一个绑定文件绑定到数据库，那么她需要 BINDADD 权限(如果这是一个新的包)或 BIND 特权(如果这是一个现有的包)。因此，假设 *nxz* 使用以下的 BIND 命令绑定这个文件：

```
BIND sampleapp.bnd QUALIFIER u1 OWNER u2 DYNAMIC RULES RUN
```

在这个例子中，所有未限定的 SQL 语句(即没有使用模式名限定其中的数据库对象的语句)会使用模式 *U1*，因为使用了 OWNER 子句，用户 *U2* 会拥有这个包。因为 *nxz* 指定了 DYNAMIC RULES RUN 子句，所以会检查运行这个应用程序(执行这个包)的用户是否具有执行动态 SQL 的特权。

但是，如果 *nxz* 使用以下的 BIND 命令绑定这个文件，那么会检查包的所有者是否具

有执行动态 SQL 的特权：

```
BIND sampleapp.bnd QUALIFIER u1 OWNER u2 DYNAMIC RULES BIND
```

在这个例子中，包的所有者被指定为 *U2*。因此，在运行时检查用户 *U2* 的特权，而不是检查运行应用程序的用户的特权。

例程特权

EXECUTE 特权应用于数据库中所有类型的例程，包括函数、过程和方法。用户一旦被授予一个例程的 EXECUTE 特权，他就可以调用这个例程，从例程创建函数(只应用于函数)，以及在任何 DDL 语句(比如 CREATE VIEW 或 CREATE TRIGGER)中引用例程。对于这个例程中访问的对象来说，不需要具有对应的特权。

13.4.6　维护特权/权限

实例级权限级别(SYSADM、SYSCTRL、SYSMAINT 和 SYSMON)和组成员关系是在 DB2 之外定义的，因此不会反映在系统编目表中。维护实例级权限需要 root 用户和 SYSADM 组成员共同完成。

DB2 将关于特权的信息存储在 7 个系统编目视图中：

- SYSCAT.DBAUTH——数据库特权
- SYSCAT.COLAUTH——表和视图列特权
- SYSCAT.INDEXAUTH——索引特权
- SYSCAT.PACKAGEAUTH——包特权
- SYSCAT.SCHEMAAUTH——模式特权
- SYSCAT.TABAUTH——表和视图特权
- SYSCAT.TBSPACEAUTH——表空间特权

可以像查询任何其他视图一样查询这些视图。例如，要查明用户 *EMMA* 拥有哪些表特权，可以发出例 13-2 所示的命令。

例 13-2　查询 SYSCAT.TABAUTH 视图来了解特权信息。

```
    SELECT substr(grantor,1,8) AS grantor,     SUBSTR(grantee,1,8) AS grantee,
           granteetype AS gtype,       SUBSTR(tabschema,1,8) AS schema,
           SUBSTR(tabname,1,8) AS tabname,      controlauth AS ctl,
           alterauth AS alt,       deleteauth AS del,       indexauth AS idx,
insertauth AS ins,  selectauth AS sel,  refauth AS ref,      updateauth AS upd
      FROM syscat.tabauth   WHERE grantee = 'EMMA'
    GRANTOR GRANTEE  GTYPE SCHEMA   TABNAME  CTL ALT DEL IDX INS SEL REF UPD
    ------- -------- ----- -------- -------- --- --- --- --- --- --- ---
```

```
    INST1    EMMA      U      INST1     TABLE1    Y   G   G   G   G   G   G   G
```

被授权者类型(GTYPE) ‘*U*’ 意味着被授权者(拥有此特权的账号)是一个用户账号，‘*G*’意味着被授权者是一个组账号。在其他列中，‘*Y*’ 意味着拥有此特权，‘*N*’ 意味着不拥有此特权，‘*G*’ 意味着拥有此特权并可以授予其他人。在例 13-2 中，可以看出用户 *EMMA* 具有表 *TABLE1* 上的 CONTROL 特权，以及所有其他可用的表特权，包括将这些特权授予其他人的能力。

注意：

隐式授予的特权——由系统授予用户的特权的授予者是 SYSIBM。

要查明具有特权的所有账号，可以查询每个系统编目视图并使用 UNION 操作符将结果组合在一起，如例 13-3 所示。

例 13-3 判断具有特权的所有授权名称。

```
    SELECT DISTINCT GRANTEE, GRANTEETYPE, 'DATABASE' FROM SYSCAT.DBAUTH
    UNION
    SELECT DISTINCT GRANTEE, GRANTEETYPE, 'TABLE   ' FROM SYSCAT.TABAUTH
    UNION
    SELECT DISTINCT GRANTEE, GRANTEETYPE, 'PACKAGE ' FROM SYSCAT.PACKAGEAUTH
    UNION
    SELECT DISTINCT GRANTEE, GRANTEETYPE, 'INDEX   ' FROM SYSCAT.INDEXAUTH
    UNION
    SELECT DISTINCT GRANTEE, GRANTEETYPE, 'COLUMN  ' FROM SYSCAT.COLAUTH
    UNION
    SELECT DISTINCT GRANTEE, GRANTEETYPE, 'SCHEMA  ' FROM SYSCAT.SCHEMAAUTH
    UNION
    SELECT DISTINCT GRANTEE, GRANTEETYPE, 'SERVER  ' FROM SYSCAT.PASSTHRUAUTH
    ORDER BY GRANTEE, GRANTEETYPE, 3
```

还可以使用控制中心查明一个用户或组具有的所有特权。操作方法是，打开控制中心，展开**所有数据库**文件夹，展开目标数据库，展开**用户和组对象**文件夹，展开**数据库用户**或**数据库组**文件夹，并双击感兴趣的用户或组的行。在图 13-15 中，打开了 *SAMPLE* 数据库中用户 *INFORMIX* 的特权对话框。选择**数据库**选项卡，会看到用户 *INFORMIX* 具有 CONNECT 和 LOAD 权限。要撤销这些权限，可以不选中权限旁边的复选框。

GET AUTHORIZATIONS 命令

DB2 有一个 GET AUTHORIZATIONS 命令，这是一个报告当前用户的特权的便捷命令。这个命令使用在数据库管理器配置文件和授权系统编目视图(SYSCAT.DBAUTH)中找到的值。发出这个命令的结果如下所示：



图 13-15　在 Control Center 中打开修改用户特权的对话框

```
db2 => get authorizations
 Administrative Authorizations for Current User
 Direct SYSADM authority                         = NO
 Direct SYSCTRL authority                        = NO
 Direct SYSMAINT authority                       = NO
 Direct DBADM authority                          = NO
 Direct CREATETAB authority                      = NO
 Direct BINDADD authority                        = NO
 Direct CONNECT authority                        = NO
 Direct CREATE_NOT_FENC authority                = NO
 Direct IMPLICIT SCHEMA authority                = NO
 Direct LOAD authority                           = NO
 Direct QUIESCE CONNECT authority                = NO
 Direct CREATE_EXTERNAL_ROUTINE authority        = NO
 Direct SYSMON authority                         = NO
 Indirect SYSADM authority                       = YES
 Indirect SYSCTRL authority                      = NO
 Indirect SYSMAINT authority                     = NO
 Indirect DBADM authority                        = NO
 Indirect CREATETAB authority                    = YES
 Indirect BINDADD authority                      = YES
 Indirect CONNECT authority                      = YES
 Indirect CREATE_NOT_FENC authority              = NO
 Indirect IMPLICIT_SCHEMA authority              = YES
 Indirect LOAD authority                         = NO
 Indirect QUIESCE_CONNECT authority              = NO
 Indirect CREATE_EXTERNAL_ROUTINE authority = NO
 Indirect SYSMON authority                       = NO
```

直接权限意味着此权限是明确授予此用户的。间接权限意味着此用户属于具有此权限

的组。如果用户被明确授予了此权限，同时又属于具有此权限的组，那么他同时具有直接和间接权限。

使用模式控制对数据库对象的访问

DB2 初级数据库管理员经常问的一个问题是，如何为用户创建适当的环境，让他们能够创建和删除自己拥有的数据库对象，同时限制其他用户访问这些对象。给每个用户提供一个他自己专用的物理数据库当然可以解决这个问题，但是这不具备可行性。最优的解决方案应该是通过使用模式来控制对数据库对象的访问。

模式是一种数据库对象，用于按照逻辑将相关的数据库对象分组。还常常用来表示对象所属权。模式具有相关联的特权，使模式所有者能够控制哪些用户有权在这个模式中创建、修改和删除对象。模式所有者最初具有这个模式上的所有特权，并能够将这些特权授予其他人。具有 SYSADM 或 DBADM 权限的用户可以修改用户在任何模式上拥有的特权。

在默认情况下，在创建数据库时，所有用户具有 IMPLICIT_SCHEMA 权限。这使任何用户可以在任何尚不存在的模式中创建对象。隐式创建的模式允许任何用户在其中创建其他对象。如果从 PUBLIC 撤销了 IMPLICIT_SCHEMA 权限，那么可以使用 CREATE SCHEMA 语句明确地创建模式，而且已经被隐式授予 IMPLICIT_SCHEMA 权限的用户(比如具有 DBADM 权限的用户)仍然可以隐式地创建模式。

为了让每个用户可以控制自己的数据库对象，数据库管理员可以为每个用户明确地创建一个模式。然后，管理员根据需要将模式上的特权授予单独的用户。这样就可以防止其他用户篡改在这个模式中创建的任何对象。为了进一步保护系统，还可以从 PUBLIC 撤销 IMPLICIT_SCHEMA 权限，这样的话，如果用户想创建数据库对象，那么必须通过他具有相应特权的模式来创建。这种方式使用户能够创建他们所需要的数据库对象，同时避免他们篡改其他用户创建的数据库对象，或者将自己和其他人创建的对象混在一起。

13.5 某银行安全规划案例

1. 任务和所需的特权/权限级别

通常，你单位中的不同用户需要不同的数据库访问级别。例如，与数据库管理员相比，客户服务代表需要更受限制的访问级别。下面我们先看有关安全的几个场景。

场景 1

小王是财务部门的一位分析师，他每天早上运行查询，查明公司的商店的收益率。在这个场景中，小王可以被授予他感兴趣的数据库上的 CONNECT 特权，以及他需要访问的

所有表上的 SELECT 特权。

场景 2

小李是一位数据库管理员，负责公司中所有数据库的维护活动。她的职责包括进行备份、在需要时恢复数据库、进行存储管理并运行跟踪。她应该不能访问数据库中的任何数据。

在这个场景中，小李可以被授予 SYSMAINT 权限。如果 SYSMAINT 太受限制了，也可以考虑授予 SYSCTRL 权限。

场景 3

小牛是一位应用程序开发人员，他负责开发和测试数据库管理器应用程序。他还可以创建包含测试数据的表。

在这个场景中，小牛需要一个或多个数据库上的 BINDADD、BIND、CONNECT 和 CREATETAB，某些特定的模式特权，以及某些表上的特权。如果他用一种外部编程语言(比如 C 或 Java)开发例程，那么可能还需要 CREATE_EXTERNAL_ROUTINE。

场景 4

小刘是营销部门的一位规划师，她在每天晚上需要将从商店收集到的新数据装载进 PRODUCT_SALES 表，从而判断新的销售趋势。

在这个场景中，小刘需要数据库上的 CONNECT 特权、LOAD 权限以及 PRODUCT_SALES 表上的 INSERT 和 SELECT 特权。

2. 安全规划

从上面的例子中我们可以看出每个场景中需要的权限和特权都不同。我们必须明确每个用户或组需要完成的任务和他需要的权限。我们需要作出一些安全规划。

在你的单位中，并非所有用户都以相同方式来划分工作职责。表 13-13 列示了某银行常见的安全相关的职务、与这些职务通常对应的任务以及完成这些任务需要的权限或特权。希望这个表能够有助于你规划安全。

表 13-13　常见职务、任务和必需的授权

职　　务	任　　务	必需的授权
部门管理员岗位	监督部门系统；创建数据库	SYSCTRL 权限。如果部门有其自己的实例，那么为 SYSADM 权限
安全管理员岗位	管理一个或多个数据库中的安全性	SECADM 权限

(续表)

职　　务	任　　务	必需的授权
数据库管理员岗位	设计、开发、操作和维护一个或多个数据库	对一个或多个数据库的 DBADM 和 SYSMAINT 权限。某些情况下，为 SYSCTRL 权限
系统操作员岗位	监视数据库并执行备份功能	SYSMAINT 权限
应用程序员岗位	开发和测试数据库管理器应用程序；也可以创建测试数据表	对现有程序包的 BINDADD、BIND，对一个或多个数据库的 CONNECT 和 CREATETAB，某些特定模式特权，以及对某些表的特权的列表。可能还需要 CREATE_EXTERNAL_ROUTINE
用户分析员岗位	通过检查系统目录视图来定义一个应用程序的数据需求	对目录视图的 SELECT；对一个或多个数据库的 CONNECT
程序最终用户	执行应用程序	对程序包的 EXECUTE；对一个或多个数据库的 CONNECT
信息中心顾问	定义查询用户的数据需求；创建表和视图，并通过授予对数据库对象的访问权来提供数据	对一个或多个数据库的 DBADM 权限
查询用户	发出 SQL 语句来检索、添加、删除或更改数据；可以将结果作为表来保存	对一个或多个数据库的 CONNECT；对要创建的表和视图的模式的 CREATEIN；以及对某些表和视图的 SELECT、INSERT、UPDATE、DELETE

13.6 执行安全审计(db2audit)

认证、权限和特权可以用来控制对数据的已知或预期存取，但是这些方法不足以阻止对数据未知或不能预测的存取。要辅助检测后面一类数据存取，DB2 提供了一种审查工具。对空闲数据访问和并发分析的成功监视可以改进对数据存取的控制，并且最后阻止恶性或无意地对数据未经授权的存取。对应用程序和个别用户存取的监视包括系统管理行为，能提供一种有关数据库系统活动的历史记录。

DB2 审查工具生成并允许 DBA 维护一系列预定义数据库事件的审计追踪。从这个工具生成的记录保存在一个审查日志文件中。对这些记录的分析可以揭示识别系统误用的使用模式。一旦识别出使用模式，就可以采取行动，来减少或者消除这些系统误用。

审计发生在实例级，这意味着审计一旦开始，它就会审计针对该实例中所有数据库的活动。审计功能可以监控不同类型的数据库事件，您可以指定只记录成功的事件还是只记录失败的事件，或者两种事件都记录。

可以使用 db2audit 命令来配置和操作审计功能。完成审计的配置并且生成了审计记录后，可以将审计记录提取到一个文本文件中，之后便可以对该文件进行分析。还可以将审计记录提取到有分隔符的 ASCII 文件中，之后可以将该文件装载到 DB2 关系表中，以便对其进行分析和查询。db2audit 命令如图 13-16 所示。

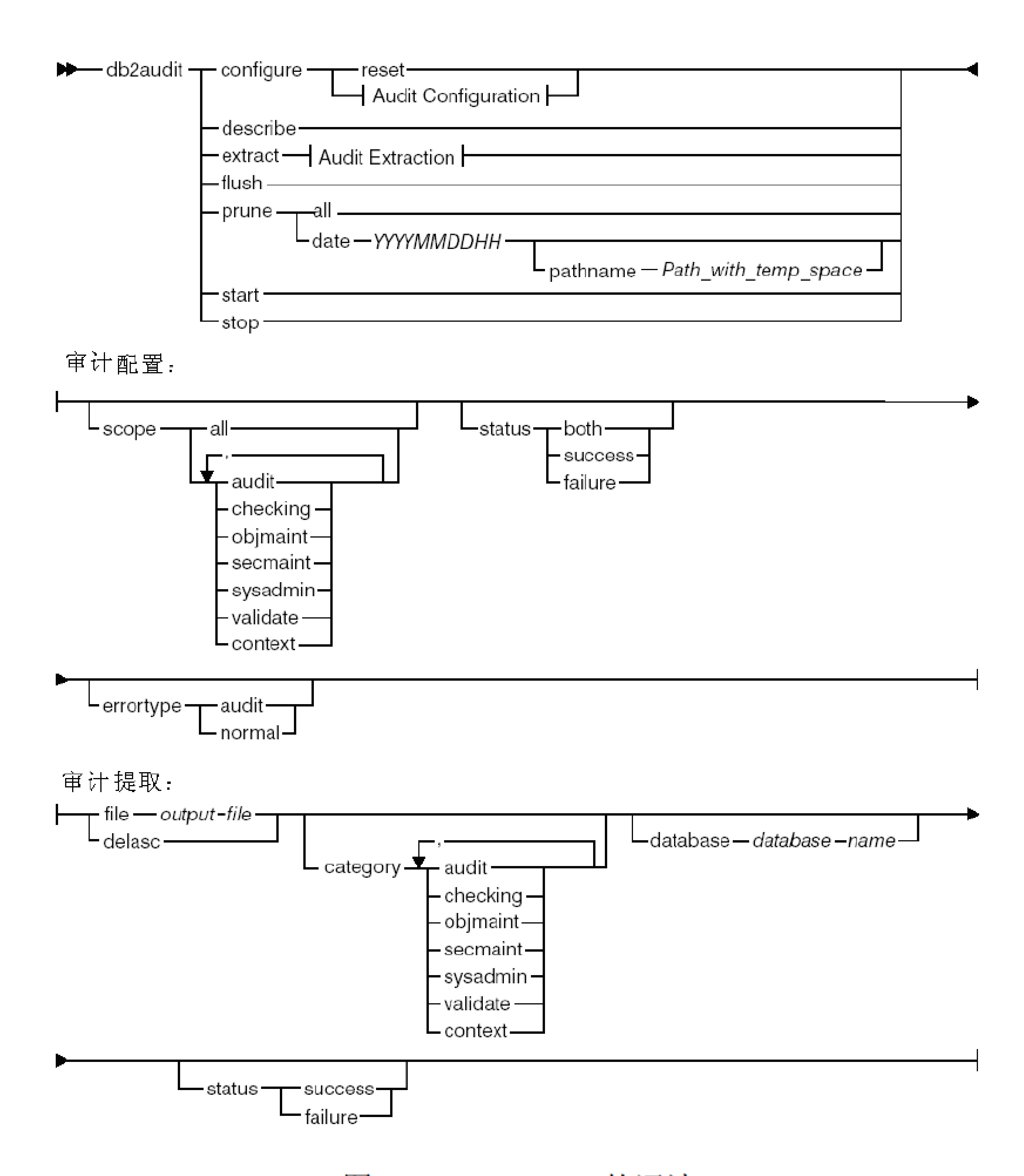


图 13-16　db2audit 的语法

下面介绍每个参数用法。

- Configure：这个参数允许在实例的安全目录中修改 db2audit.cfg 配置文件。对这个文件的更新发生在关闭实例的时候。实例激活时发生的更新动态影响着 DB2 在所有分区上进行的审查。如果已经启动了审查工具，并且正在审查可审查的事件的审查类目，那么配置文件上的配置行为就会导致创建审查记录。下面给出配置文件上可能的配置行为：
  - ◊ RESET。这个行为引起配置文件恢复初始配置，其中 SCOPE 表示所有类别(除了 CONTEXT)，STATUS 表示 FAILURE，ERRORTYPE 表示 NORMAL，而审查工具为 OFF。如果原始文件丢失或损坏了，那么这就将创建一个新的审查配置文件。
  - ◊ SCOPE：这个行为指定审查哪个类别的事件。这个行为也允许特别关注审查和减少日志的增长。推荐日志的事件类型和数目尽可能少，否则认证日志就会快速增长。审查可行的事件类目有：
    - 审查(AUDIT)：改变审查设置或存取审查日志时生成记录；
    - 认证检查(CHECKING)：试图存取的认证检查或对 DB2 对象或函数进行操作时生成记录；
    - 对象维护(OBJMAINT)：创建或删除数据对象时生成记录；
    - 安全维护(SECMAINT)：授予或撤回对象或数据库特权时生成记录。当修改数据库管理器安全配置参数 SYSADM_GROUP、SYSCTRL_GROUP 或 SYSMAINT_GROUP 时也生成记录；
    - 系统管理(SYSADMIN)：当执行需要 SYSADM、SYSMAINT 或 SYSCTRL 权限的操作系统时生成记录；
    - 用户确认(VALIDATE)：认证用户或再次获取系统安全信息时生成记录；
    - 操作上下文(CONTEXT)：当执行一个数据库操作时生成记录来显示操作的上下文。这个种类允许审查日志文件更好的解释。
    - 当一组事件用于日志事件的相关区域时，这组事件可以与一个单数据库操作相关联。例如，动态 SQL 的一条 SQL 语句、静态 SQL 的包标识符或执行的操作类型的指示器，例如，CONNECT 可以在分析审查结果时提供必要的上下文。
  - ◊ STATUS：这个行为指定是否只有成功或失败事件或者成功和失败事件被日志。
  - ◊ ERRORTYPE：AUDIT 和 NORMAL。

- Describe：这个参数描述标准输出当前审查配置信息和状态。
- Extract：这个参数允许从审查日志移除审查记录到一个预示的目的地。如果没有指定可选的子句，那么所有审查记录都被提取和放到一个文本 reportfile 中。如果没有指定“extract”参数，那么审查记录就继续放到安全目录的 db2audit.log 文件中。如果已经存在输出文件，那么就返回一个错误消息。

下面给出用于提取的可能选项。

- FILE：提取的审查记录放到一个文本文件中(output_file)。
- DELASC：提取的审查记录放到限定的适于导入 DB2 关系表的 ASCII 格式文件中。

现在我们举一个关于审计的例子。假如，假设您从某个机房维护用户那里得到匿名举报，说有一个名为“王二”的用户正在试图更新他原本无权访问的数据库对象和表(存放个人工资信息的表)的访问权。于是您决定审计监控 DB2 实例，以期发现失败的授权验证尝试。

您首先对审计功能进行配置，使之审计 CHECKING 事件类型，只记录失败的尝试，并且使用 NORMAL 错误处理：

```
db2audit configure scope checking status failure errortype normal
```

完成配置后，启动审计功能：

```
db2audit start
```

在审计期间，王二来到数据库服务器，并完成登录。他打开一个命令行窗口，连接到 SAMPLE 数据库，并尝试更新 EMPLOYEE 表中的雇员工资(当然这样的尝试会遭到失败)。他发出以下 SQL 语句：

```
connect to sample user 'wanger' using password
update tedwas.employee set salary = salary * 1.5
```

于是收到以下错误消息：

```
    DB21034E The command was processed as an SQL statement because it was not
avalid Command Line Processor command. During SQL processing it returned:
    SQL0551N "wanger" does not have the privilege to perform operation "UPDATE"
onobject "TEDWAS.EMPLOYEE". SQLSTATE=42501
```

表明他没有更新那个表的许可，他快速退出服务器，并离开现场，自以为没有人注意到这一切。

一个小时过去了，您决定检查审计日志的内容。于是将 db2audit.log 文件中的记录提取到带分隔符的 ASCII 文件中：

```
    db2audit extract delasc delimiter ; category checking database sample status
failure
```

为了让之前创建的 DB2 表保存审计数据，使用以下命令将从 checking.del 文件中提取的数据装载到 CHECKING 表中：

```
    LOAD FROM checking.del OF del MODIFIED BY CHARDEL; INSERT INTO audit.checking
```

您可以查询 AUDIT.CHECKING 表，以发现关于失败的授权尝试的更多信息：

```
    SELECT category、event、appid、appname、userid、authid FROM audit.checking
```

在例 13-4 显示的查询结果中，您可以看到有一条关于失败的更新语句的审计记录。

例 13-4 查询 CHECKING 表的结果。

```
    SELECT category、event、appid、appname、userid、authid FROM audit.checking
    CATEGORY EVENT              APPID                  APPNAME        AUTHID
    ---------------------  ---------------------  -------------  -------
    CHECKING CHECKING_OBJECT  *LOCAL.DB2.080206220334   db2bp.exe       wanger
    1 record(s) selected.
```

这个输出可以证实“王二”曾经尝试访问他无权访问的一个表。现在，您的怀疑得到了证实，您可以继续收集更多的证据提交给他的领导，以便他们采取纠正措施。

DB2 的审计功能非常强大，可以为您提供审计访问尝试时所需的详细信息。虽然对未预见到的事件进行被动的监控是不可避免的，但是您应该使前摄性的审计成为安全计划中的重要组成部分。但是也要注意安全审计对性能有些许影响。

13.7 基于标签的访问控制(LBAC)及案例

DB2 V9 中新增的一个概念是基于标签的访问控制(LBAC)。LBAC 为 DBA 提供了在表的行或列级限制读/写特权的能力。在以前，进行这种限制的唯一方法是创建一个视图，授权用户使用这个视图，并撤销对基表的访问权。

基于标签的访问控制(LBAC)使安全性管理员能够准确地确定对于各行各列具有写访问权的用户和具有读访问权的用户。安全性管理员通过创建安全策略来配置 LBAC 系统。安全策略描述的是用来确定哪些用户能够访问哪些数据的条件。对于任何一个表而言，只能使用一个安全策略来保护它，但不同的表可以由不同的安全策略保护。

创建安全策略之后，安全性管理员将创建称为安全标签和免除权的数据库对象，这些对象是安全策略的组成部分。安全标签描述一组安全条件。免除权遵循一个规则，即在拥

有免除权的用户访问受安全策略保护的数据时，不需要强制对该用户比较安全标签。

一旦创建了安全标签，就可以使其与各个表列和表行相关联，从而保护存放在那些位置的数据。受安全标签保护的数据称为受保护数据。安全性管理员通过将安全标签授予用户来允许该用户访问受保护数据。当用户尝试访问受保护数据时，该用户的安全标签将与用于保护该数据的安全标签进行比较。用于保护该数据的标签将阻塞一部分用户的安全标签。

LBAC 由安全管理员通过创建安全策略来设置。每个表只能由一个安全策略来控制，但是系统中可以有任意数量的安全策略。设置 LBAC 需要几个步骤。必须做的第一件事情是—— 决定对于您的数据需要什么类型的访问控制。下面我们举个例子：

我们做出以下假设。在您的单位有三类人，如表 13-14 所示。

表 13-14　职员及其角色

名　　称	在组织中的角色
Jane	人力资源执行官
Joe	D11 和 E21 部门的经理
Frank	团队主管(A00 部门)

现在，在单位的数据库中有一个定义职员信息的表。这个表类似于 SAMPLE 数据库中的 EMP 表。它包含关于职员和他们所属的部门的数据。它现在的定义如下：

```
db2 => describe select * from emp
 sqltype              sqllen  sqlname.data                    sqlname.length
 -------------------- ------  ------------------------------  --------------
 452   CHARACTER           6  EMPNO                                        5
 448   VARCHAR            12  FIRSTNME                                     8
 453   CHARACTER           1  MIDINIT                                      7
 448   VARCHAR            15  LASTNAME                                     8
 453   CHARACTER           3  WORKDEPT                                     8
 453   CHARACTER           4  PHONENO                                      7
 385   DATE               10  HIREDATE                                     8
 453   CHARACTER           8  JOB                                          3
 500   SMALLINT            2  EDLEVEL                                      7
 453   CHARACTER           1  SEX                                          3
 385   DATE               10  BIRTHDATE                                    9
 485   DECIMAL          9, 2  SALARY                                       6
 485   DECIMAL          9, 2  BONUS                                        5
 485   DECIMAL          9, 2  COMM                                         4
```

单位会定期对规则进行审计。审计指出，职员不应该能够访问机密的数据。规则规定，执行官对所有职员记录有完全的读/写访问权，经理对自己部门的职员记录有读/写访问权，而团队主管只能读取部门中他们领导的职员的记录。

我们要设置 LBAC 安全策略来实现这些规则。

(1) 定义安全策略和标签，并将安全标签授予用户。

(2) 在 EMP 表中添加安全标签列并将安全策略连接到它。

定义安全策略和标签

为了定义安全策略和标签，需要 SECADM 权限。

(1) 创建安全标签组件

首先，需要决定对于这个策略来说最合适的安全组件类型。在这个示例中，最合适的策略类型是“TREE”。TREE 策略意味着可以定义一组标签，让子组件拥有它们的父组件的权限的子集。在这个示例中，创建一个名为“J_DEPT”的安全组件。

```
CREATE SECURITY LABEL COMPONENT J_DEPT
       TREE('HR_EXECUTIVE' ROOT,
            'MAN_D11_E21' UNDER 'HR_EXECUTIVE'
            'A00' UNDER 'HR_EXECUTIVE',
            'B01' UNDER 'HR_EXECUTIVE',
            'C01' UNDER 'HR_EXECUTIVE',
            'D11' UNDER 'MAN_D11_E21',
            'D21' UNDER 'HR_EXECUTIVE',
            'E01' UNDER 'HR_EXECUTIVE',
            'E11' UNDER 'HR_EXECUTIVE',
            'E21' UNDER 'MAN_D11_E21'       )
```

上面的布局说明根是 HR_EXECUTIVE，这个执行官领导的所有部门都是它的子组件。

(2) 定义安全策略

在上面的示例中，设置 LBAC 所需的下一个步骤是定义与上面的安全标签组件相关联的策略。一个安全策略可以使用多个组件。

```
CREATE SECURITY POLICY J_DEPT_POLICY
     COMPONENTS J_DEPT
     WITH DB2LBACRULES
     RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
```

(3) 创建安全标签

设置安全策略的第三步是创建安全标签。在这里将指定每个用户具有的不同角色。因为这个示例非常简单，只有 3 个标签：Executive、Manager 和 Team Lead。

```
        CREATE SECURITY LABEL J_DEPT_POLICY.EXECUTIVE
              COMPONENT J_DEPT 'HR_EXECUTIVE'
        CREATE SECURITY LABEL J_DEPT_POLICY.MANAGE_D11_E21
              COMPONENT J_DEPT 'MAN_D11_E21'
        CREATE SECURITY LABEL J_DEPT_POLICY.A00
              COMPONENT J_DEPT 'A00'
        CREATE SECURITY LABEL J_DEPT_POLICY.B01
              COMPONENT J_DEPT 'B01'
        CREATE SECURITY LABEL J_DEPT_POLICY.C01
              COMPONENT J_DEPT 'C01'
        CREATE SECURITY LABEL J_DEPT_POLICY.D11
              COMPONENT J_DEPT 'D11'
        CREATE SECURITY LABEL J_DEPT_POLICY.D21
              COMPONENT J_DEPT 'D21'
        CREATE SECURITY LABEL J_DEPT_POLICY.E01
              COMPONENT J_DEPT 'E01'
        CREATE SECURITY LABEL J_DEPT_POLICY.E11
              COMPONENT J_DEPT 'E11'
        CREATE SECURITY LABEL J_DEPT_POLICY.E21
              COMPONENT J_DEPT 'E21'
```

在下一步中，将定义与这些标签相关联的实际权限。

(4) 根据标签授予权限

下面的步骤描述了对表数据授予权限的过程。权限可以是 ALL ACCESS、WRITE ACCESS 或 READ ACCESS。如果一个用户没有获得上述的任何权限，那么这个用户就不能访问任何表数据。请记住，执行官有完全的访问权，经理对自己的部门有完全的访问权，而团队主管只对他们领导的部门成员有读访问权。

```
db2 grant security label J_DEPT_POLICY.A00 to user Frank for read access
db2 grant security label J_DEPT_POLICY.MANAGE_D11_E21 to user Joe for all access
db2 grant security label J_DEPT_POLICY.EXECUTIVE to user Jane for all access
```

为用户设置以上标签，根据步骤(1)中的树定义来分配权限。因为用户 Joe 被标为 MANAGE_D11_E21 并获得所有权限，所以他将能够读写那些安全标记为 J_DEPT_POLICY.D11 或 J_DEPT_POLICY.E21 的行(因为它们是他的子组件)。

修改 EMP 表

在修改 EMP 表时，必须创建一个额外的列来存储安全标签。这个列的类型是“DB2SECURITYLABEL”。您可以修改 SAMPLE 数据库中现有的 EMP 表。为此，必须使用在这个策略中被授予根级特权的用户，在这个示例中就是用户 Jane。

```
CONNECT TO SAMPLE USER Jane USING password
ALTER TABLE EMP ADD COLUMN DEPT TAG DB2SECURITYLABEL
  ADD SECURITY POLICY J_DEPT_POLICY
```

如果从 EMP 表进行选择，就会看到刚才定义的新列。由于是用在 EXECUTIVE 级上定义的用户执行这一修改，所以添加的所有安全标记都是 EXECUTIVE。为了改变这一情况，需要更新这个表。

```
db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
varchar(SECLABEL TO CHAR('J DEPT POLICY',DEPT TAG),30) from gmilne.emp
EMPNO  FIRSTNME     LASTNAME        WORKDEPT SALARY      6
------ ------------ --------------- -------- ----------- --------------
000010 CHRISTINE    HAAS            A00       152750.00 HR_EXECUTIVE
000020 MICHAEL      THOMPSON        B01        94250.00 HR EXECUTIVE
000030 SALLY        KWAN            C01        98250.00 HR_EXECUTIVE
000050 JOHN         GEYER           E01        80175.00 HR EXECUTIVE
000060 IRVING       STERN           D11        72250.00 HR_EXECUTIVE
000070 EVA          PULASKI         D21        96170.00 HR EXECUTIVE
000090 EILEEN       HENDERSON       E11        89750.00 HR_EXECUTIVE
000100 THEODORE     SPENSER         E21        86150.00 HR EXECUTIVE
……………………………节省篇幅，略去部分数据……………………………………………………
200340 ROY          ALONZO          E21        31840.00 HR EXECUTIVE
  42 record(s) selected.
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','A00')) where
WORKDEPT='A00'
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','B01')) where
WORKDEPT='B01'
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','C01')) where
WORKDEPT='C01'
    update emp set DEPT_TAG=(SECLABEL_BY_NAME('J_DEPT_POLICY','D11')) where
WORKDEPT='D11'
    update emp set DEPT_TAG=(SECLABEL_BY_NAME('J_DEPT_POLICY','D21')) where
WORKDEPT='D21'
    update emp set DEPT_TAG=(SECLABEL_BY_NAME('J_DEPT_POLICY','E01')) where
WORKDEPT='E01'
    update emp set DEPT_TAG=(SECLABEL_BY_NAME('J_DEPT_POLICY','E11')) where
WORKDEPT='E11'
    update emp set DEPT_TAG=(SECLABEL_BY_NAME('J_DEPT_POLICY','E21')) where
WORKDEPT='E21'
db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
varchar(SECLABEL_TO_CHAR('J_DEPT_POLICY',DEPT_TAG),30) from emp
```

```
EMPNO  FIRSTNME     LASTNAME        WORKDEPT SALARY      6
------ ------------ --------------- -------- -----------
000010 CHRISTINE    HAAS            A00      152750.00 A00
000020 MICHAEL      THOMPSON        B01       94250.00 B01
000030 SALLY        KWAN            C01       98250.00 C01
000050 JOHN         GEYER           E01       80175.00 E01
000060 IRVING       STERN           D11       72250.00 D11
.........................节省篇幅，略去部分数据.........................
200310 MICHELLE     SPRINGER        E11       35900.00 E11
200330 HELENA       WONG            E21       35370.00 E21
200340 ROY          ALONZO          E21       31840.00 E21
  42 record(s) selected.
```

在更新之后，我们来看看各个用户能够做什么。使用 Executive 用户 ID Jane 连接数据库。首先执行与前面一样的选择语句：

```
 db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
varchar(SECLABEL_TO_CHAR('J_DEPT_POLICY',DEPT_TAG),30) from gmilne.emp
EMPNO  FIRSTNME     LASTNAME        WORKDEPT SALARY      6
------ ------------ --------------- -------- -----------
000010 CHRISTINE    HAAS            A00      152750.00 A00
000020 MICHAEL      THOMPSON        B01       94250.00 B01
000030 SALLY        KWAN            C01       98250.00 C01
000050 JOHN         GEYER           E01       80175.00 E01
.........................节省篇幅，略去部分数据.........................
200330 HELENA       WONG            E21       35370.00 E21
200340 ROY          ALONZO          E21       31840.00 E21
  42 record(s) selected.
```

以及更新命令：

```
db2 => update gmilne.emp set DEPT_TAG=(SECLABEL_BY_NAME('J_DEPT_POLICY','E01'))
where WORKDEPT='E01' DB20000I The SQL command completed successfully.
```

可以看到，Jane 对表中的所有数据有完全的访问权。现在，看看 Joe 可以看到的内容。首先进行选择：

```
db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
varchar(SECLABEL_TO_CHAR('J_DEPT_POLICY',DEPT_TAG),30) from gmilne.emp
EMPNO  FIRSTNME     LASTNAME        WORKDEPT SALARY      6
------ ------------ --------------- -------- -----------
000060 IRVING       STERN           D11       72250.00 D11
000100 THEODORE     SPENSER         E21       86150.00 E21
000150 BRUCE        ADAMSON         D11       55280.00 D11
```

```
000160 ELIZABETH    PIANKA          D11          62250.00 D11
000170 MASATOSHI    YOSHIMURA       D11          44680.00 D11
000180 MARILYN      SCOUTTEN        D11          51340.00 D11
000190 JAMES        WALKER          D11          50450.00 D11
000200 DAVID        BROWN           D11          57740.00 D11
000210 WILLIAM      JONES           D11          68270.00 D11
000220 JENNIFER     LUTZ            D11          49840.00 D11
000320 RAMLAL       MEHTA           E21          39950.00 E21
000330 WING         LEE             E21          45370.00 E21
000340 JASON        GOUNOT          E21          43840.00 E21
200170 KIYOSHI      YAMAMOTO        D11          64680.00 D11
200220 REBA         JOHN            D11          69840.00 D11
200330 HELENA       WONG            E21          35370.00 E21
200340 ROY          ALONZO          E21          31840.00 E21
  17 record(s) selected.
```

看到了吗？他只能看到 D11 和 E21 部门的信息。如果他试图选择不允许他访问的表数据，那么会发生什么：

```
db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
varchar(SECLABEL TO CHAR('J DEPT POLICY',DEPT TAG),30)
from gmilne.emp where empno='000130'
EMPNO  FIRSTNME     LASTNAME        WORKDEPT SALARY      6
------ ------------ --------------- -------- -----------
  0 record(s) selected.
```

在前面 Jane 进行选择的结果中我们看到，有一个职员的 empno 是 000130，但是现在却不允许 Joe 看到它。

下面是最后一个测试，对用户 Frank 的测试。

首先，运行与前两个用户相同的选择：

```
db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
varchar(SECLABEL TO CHAR('J DEPT POLICY',DEPT TAG),30) from gmilne.emp
EMPNO  FIRSTNME     LASTNAME        WORKDEPT SALARY      6
------ ------------ --------------- -------- -----------
000010 CHRISTINE    HAAS            A00         152750.00 A00
000110 VINCENZO     LUCCHESSI       A00          66500.00 A00
000120 SEAN         O'CONNELL       A00          49250.00 A00
200010 DIAN         HEMMINGER       A00          46500.00 A00
200120 GREG         ORLANDO         A00          39250.00 A00
  5 record(s) selected.
```

在这里可以看到，Frank 只能看到部门中他领导的用户的相关信息。我们来看看在他尝试进行更新时会发生什么：

```
    db2 => update gmilne.emp set
DEPT_TAG=(SECLABEL_BY_NAME('J_DEPT_POLICY','A00'))
    where WORKDEPT='A00'DB21034E  The command was processed as an SQL statement
    because it was not a valid Command Line Processor command.  During SQL
processing it
    returned:
    SQL20402N AuthorizationID"FRANK" does not have the LBAC credentials to
    perform the "UPDATE" operation on table "EMPLOYEE".  SQLSTATE=42519
```

尽管他尝试更新的记录是在自己的部门中的，但是访问安全策略只允许他对表进行读访问。可以看出，我们的业务需求已经得到了满足。

13.8　本章小结

在本章我们讲解了 DB2 中定义的各种权限级别和特权，以及如何使用命令行语法和控制中心将它们分配给用户。还讨论了特权的某些细节，包括隐式特权、静态和动态 SQL 之间授权方式的差异，以及特权信息如何存储在系统编目表中。最后，讨论了在多用户环境中如何使用模式有效地控制对数据库对象的访问、如何有效地规划安全。有了这些知识后，您就应该能够定义一个特权/权限策略，防止用户意外或故意地威胁系统的安全。